A Hybrid Algorithm for the Unrelated Parallel Machine Scheduling Problem

Marcelo Ferreira Rego[1,2(✉)] and Marcone Jamilson Freitas Souza[1]

[1] Programa de Pós-Graduação em Ciência da Computação,
Universidade Federal de Ouro Preto (UFOP),
Ouro Preto, Minas Gerais 35.400-000, Brazil
marcone@ufop.edu.br

[2] Universidade Federal dos Vales do Jequitinhonha e Mucuri (UFVJM),
Diamantina 39.100-000, Brazil
marcelofr@ufvjm.edu.br

Abstract. This work proposes a hybrid algorithm for the unrelated parallel machine scheduling problem with sequence-dependent setup times, aiming to minimize the makespan. The proposed algorithm, named Enhanced Smart General Variable Neighborhood Search (e-SGVNS), combines heuristic and exact optimization strategies to explore the solution space of the problem. The exact strategy works like a local search and consists of applying a mathematical programming formulation based on the time-dependent traveling salesman problem to obtain the optimal solution to the sequencing problem on each machine. In turn, the heuristic strategy explores neighborhoods based on swap and insertion moves. The computational results, performed in benchmark instances from literature, showed that e-SGVNS is competitive when compared to state-of-the-art algorithms.

Keywords: Unrelated parallel machine scheduling · Makespan · VNS · Metaheuristic · Mixed integer linear programming

1 Introduction

The Unrelated Parallel Machine Scheduling Problem with Setup Times (UPMSP-ST) consists of scheduling a set N of n independent jobs on a set M of m unrelated parallel machines. Each job $j \in N$ must be processed exactly once by only one machine $i \in M$, and requires a processing time p_{ij}. Each machine can process only one job at a time. In addition, job execution requires a setup time S_{ijk}, which depends on the machine i and the sequence in which jobs j and k will be processed. The objective is to minimize the makespan.

The study of the UPMSP-ST is relevant due to its theoretical and practical importance. From a theoretical point of view, it attracts the interest of researchers because it is NP-hard, since it is a generalization of the Parallel Machine Scheduling Problem with Identical Machines [1]. In practical, it is found in a large number of industries, such as the textile industry [2]. According to

J. Filipe et al. (Eds.): ICEIS 2019, LNBIP 378, pp. 37–56, 2020.
https://doi.org/10.1007/978-3-030-40783-4_3

Avalos-Rosales et al. [3], in a lot of situations where there are different production capacities, the setup time of machine depends on the previous job to be processed [4]. This situation is also found in the manufacture of chemical products, where the reactors must be cleaned between the handling of two mixtures; however, the time required for cleaning depends on the jobs that were previously completed [5].

In this work, a hybrid algorithm, named e-SGVNS, is proposed. It is an improvement of the SGVNS algorithm from Rego and Souza [6]. It is based on the General Variable Neighborhood Search – GVNS [7] and explores the solution space through five strategies: swap of jobs in the same machine; insertion of job on the same machine; swap of jobs between machines; insertion of jobs on different machines; and an application of a mathematical programming formulation based on the time-dependent traveling salesman problem to get the optimal solution to the sequencing problem on each machine. The first four strategies are used as shaking mechanism, while the last three are applied as local search through the Variable Neighborhood Descent. Unlike SGVNS, the proposed algorithm limits the increase of perturbation level. In addition, it applies MILP to all machines whose completion time is equal to makespan and not just to a single machine that meets this condition. This algorithm has been shown to be competitive when compared to state-of-the-art algorithms.

The remainder of this paper is organized as follows: Sect. 2 gives a brief review of the literature. In Sect. 3, a mathematical programming formulation for the problem is presented. In Sect. 4, the proposed algorithm is detailed. The results are presented in Sect. 6, while in Sect. 7 the work is concluded.

2 Related Work

Santos et al. [8] implemented four different stochastic local search (SLS) methods for the UPMSP-ST. The algorithms explore six different neighborhoods. The computational results show that the SLS algorithms produce good results, outperforming the current best algorithms for the UPMSP-ST. They updated 901 best-known solutions from 1000 instances used for testing.

Arnaout [9] introduced and applied a Worm Optimization (WO) algorithm for the UPMSP-ST. The WO algorithm is based on the behaviors of the worm, which is a nematode with only 302 neurons. The WO algorithm was compared to tabu search (TS), ant colony optimization (ACO), restrictive simulated annealing (RSA), genetic algorithm (GA), and ABC/HABC. The experiments showed the superiority of WO, followed by HABC, ABC, RSA, GALA, ACO, and TS last.

Arnaout et al. [10] proposed a two-stage Ant Colony Optimization algorithm (ACOII) for the UPMSP-ST. This algorithm is an enhancement of the ACOI algorithm that was introduced in [11]. An extensive set of experiments was performed to verify the quality of the method. The results proved the superiority of the ACOII in relation to the other algorithms with which it was compared.

Tran et al. [5] introduced a new mathematical formulation for the UPMSP-ST. This formulation provides dual bounds that are more efficient to find the

optimum solution. The computational experiments showed that it is possible to solve larger instances than it was possible to solve with other previously existing formulations.

A variant of the Large Neighborhood Search metaheuristic, using Learning Automata to adapt the probabilities of using removal and insertion heuristics and methods, named LA-ALNS, is presented by Cota et al. [12] for the UPMSP-ST. The algorithm was used to solve instances of up to 150 jobs and 10 machines. The LA-ALNS was compared with three other algorithms and the results show that the developed method performs best in 88% of the instances. In addition, statistical tests indicated that LA-ALNS is better than the other algorithms found in the literature.

The UPMSP-ST was also approached by Fanjul-Peyro and Ruiz [13]. Seven algorithms were proposed: IG, NSP, VIR, IG+, NSP+, VIR+ and NVST-IG+. The first three are the base algorithms. The following three are improved versions of these latest algorithms. Finally, the last algorithm is a combination of the best ideas from previous algorithms. These methods are mainly composed of a solution initialization, a Variable Neighborhood Descent – VND method [7] and a solution modification procedure. Tests were performed with 1400 instances and it was showed that the results were statistically better than the algorithms previously considered state-of-the-art, which were, [14,15].

A Genetic Algorithm was proposed by Vallada and Ruiz [16] for the UPMSP-ST. The algorithm includes a fast local search and a new crossover operator. Furthermore, the work also provides a mixed integer linear programming model for the problem. After several statistical analyzes, the authors concluded that their method provides better results for small instances and, especially, for large instances, when compared with other methods of the literature at the time [17,18].

Rego and Souza [6] proposed the SGVNS algorithm for treating the UPMSP-ST. It explores the solution space by three strategies of local search: insertion of jobs in different machines, swap of jobs between machines and an application of a mixed integer linear programming formulation to obtain optimum scheduling on each machine. The SGVNS algorithm was tested in 810 instances and compared to four other literature methods (ACOII, AIRP and LA-ALNS). SGVNS had better performance when executed in small instances. The results of LA-ALNS and ACOII were significantly better than the results of SGVNS algorithm. Even so, SGVNS was superior in 5 groups of instances and able to find best results in 79 of the 810 instances.

3 Mathematical Formulation

This section provides a Mixed Integer Linear Programming (MILP) formulation for the unrelated parallel machine scheduling problem with sequence-dependent setup times with the objective of minimizing the makespan. This formulation was proposed by Tran et al. [5].

In order to introduce this MILP, the parameters and decision variables are defined and shown in Table 1.

The objective function is given by Eq. (1):

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

and the constraints are given by Eqs. (2)–(10):

$$\sum_{i \in M} \sum_{\substack{j \in N \cup \{0\}, \\ j \neq k}} X_{ijk} = 1 \qquad \forall k \in N \tag{2}$$

$$\sum_{i \in M} \sum_{\substack{k \in N \cup \{0\}, \\ j \neq k}} X_{ijk} = 1 \qquad \forall j \in N \tag{3}$$

$$\sum_{\substack{k \in N \cup \{0\}, \\ k \neq j}} X_{ijk} = \sum_{\substack{h \in N \cup \{0\}, \\ h \neq j}} X_{ihj} \qquad \forall j \in N,\ \forall i \in M \tag{4}$$

$$C_k \geqslant C_j + S_{ijk} + p_{ik} - V(1 - x_{ijk}) \qquad \forall j \in N \cup \{0\}, \forall k \in N, j \neq k, \forall i \in M \tag{5}$$

$$\sum_{j \in N} X_{i0j} \leqslant 1 \qquad \forall i \in M \tag{6}$$

$$C_0 = 0 \tag{7}$$

$$\sum_{\substack{j \in N \cup \{0\}, \\ j \neq k}} \sum_{k \in N} (S_{ijk} + p_{ik}) X_{ijk} = O_i, \qquad \forall i \in M, \tag{8}$$

$$O_i \leqslant C_{\max}, \qquad \forall i \in M, \tag{9}$$

$$X_{ijk} \in \{0, 1\} \qquad \forall j \in N \cup \{0\}, \forall k \in N, j \neq k, \forall i \in M, \tag{10}$$

$$C_j \geq 0 \qquad \forall j \in N \tag{11}$$

$$O_i \geq 0 \qquad \forall i \in M \tag{12}$$

$$C_{\max} \geq 0 \tag{13}$$

Equation (1) defines the objective function of the problem, which is to minimize the maximum completion time or makespan. Eqs. (2)–(10) define the constraints of the model. The constraint set (2) ensures that each job is assigned to exactly one machine and has exactly one predecessor job. Constraints (3) define

Table 1. Parameters and decision variables of Tran et al. [5] model.

Name	Description	Type
V	A very large constant	Parameter
N	Set of jobs	
M	Set of machines	
p_{jk}	Processing time of job j on machine i	
S_{ijk}	Setup time required for processing job $k \in N$ immediately after job $j \in N$ on machine $i \in M$	
X_{ijk}	Equal to 1, if job j immediately precedes job k on machine i and 0, otherwise	Decision variable
C_j	Completion time of job j	
O_i	Completion time of last job in machine i	
$C_{\max}$	Maximum completion time	

that every job has exactly one successor job. Each constraint (4) establishes that if a job j is scheduled on a machine i, then a predecessor job h and a successor job k must exist in the same machine. Constraints (5) ensure a right processing order. Basically, if a job k is assigned to a machine i immediately after job j, that is, if $X_{ijk} = 1$, the completion time C_k of this job k) must be greater than or equal to the completion time C_j of job j, added to setup time between jobs j and k and the processing time p_{ik} of k on machine i. If $X_{ijk} = 0$, then a sufficiently high value V makes this constraint redundant. With constraint set (6) we define at most one job is scheduled as the first job on each machine. Constraints (7) establish that the completion time of the dummy job is zero. Constraints (8) compute, for each machine, the time it finishes processing its last job. Constraints (9) define the maximum completion time. Constraints (10)–(13) define the domain of the decision variables.

4 The Enhanced Smart GVNS Algorithm

The algorithm presented in this work, named e-SGVNS, is a is an improvement of the SGVNS algorithm from Rego and Souza [6]. In turn, SGVNS is a variant of the General Variable Neighborhood Search (GVNS) metaheuristic [7].

This metaheuristic performs systematic neighborhood exchanges to explore the solution space of the problem. It uses the Variable Neighborhood Descent procedure – VND [19], described in Sect. 4.4 as the local search procedure, and it has a perturbation phase in order to not get stuck in local optima, which is described in Sect. 4.2.

The perturbation phase of e-SGVNS depends of the perturbation level of the algorithm. This level is always increased when a certain number of VND applications occur without producing improvement in the current solution. The e-SGVNS was implemented according to the Algorithm 1:

Algorithm 1. e-SGVNS.

```
   input : stopping criterion, MaxP, MaxSameLevelP, 𝒩
 1 s_0 ← Initial Solution();
 2 ItSameLevelP ← 1;
 3 p ← 2;
 4 s ← VND(s_0, 𝒩);
 5 while (stopping criterion was not satisfied) do
 6 |   s' ← Shaking(s, p);
 7 |   s'' ← VND(s', 𝒩);
 8 |   if (f(s'') < f(s)) then
 9 |   |   s ← s'';
10 |   |   p ← 2;
11 |   |   ItSameLevelP ← 1;
12 |   end
13 |   else
14 |   |   ItSameLevelP ← ItSameLevelP + 1;
15 |   |   if (ItSameLevelP > MaxSameLevelP) then
16 |   |   |   p ← p + 1;
17 |   |   |   ItSameLevelP ← 1;
18 |   |   |   if p > MaxP then
19 |   |   |   |   p ← 2;
20 |   |   |   end
21 |   |   end
22 |   end
23 end
24 return s ;
```

Algorithm 1 has the following inputs: (1) the stopping criterion, which in our case was the CPU timeout t, described in Sect. 5.2; (2) $MaxP$, maximum level of perturbation; (3) $MaxSameLevelP$, the maximum number of iterations without improvement in $f(s)$ with the same perturbation level; (4) the set $\mathcal{N}$ of neighborhoods. In line 1, the solution s is initialized from the solution obtained by the procedure defined in Sect. 4.1. In line 6, a random neighbor s' is generated from a perturbation performed according to the procedure defined in Sect. 4.2. The loop from lines 5–23 is repeated while the stopping criterion is not satisfied. In line 7, a local search on s' using the neighborhood structures described in Sect. 4.3 is performed. It stops when it finds the first solution that is better than s or when the whole neighborhood has been explored. The solution returned by this local search is attributed to s'' if its value is better than the current solution. Otherwise, the procedure continues to exploit from a new neighborhood structure.

4.1 Initial Solution

An initial solution to the problem is constructed according to Algorithm 2.

Algorithm 2. Initial Solution.

```
   input : M, N
1  foreach k ∈ N do
2  |  Find the machine i and the position j for the job k that produces the lowest
   |  cost for the objective function;
3  |  Insert job k in position j on machine i;
4  end
```

Algorithm 2 gets as input the sets M and N of machines and jobs, respectively. At each iteration, it looks for position j on a machine i to insert the job k into scheduling, it always chooses the position that gives the smallest increase in the objective function according to Eq. (1). The previously described steps are repeated for all jobs, so the procedure ends when all jobs are already allocated on some machine.

4.2 Shaking

The shaking procedure is an important phase of a VNS-based algorithm. It is applied to not limit the local search to the same region of the solution space of the problem, and consequently explore other solutions. The shaking procedure implemented increases progressively the level of perturbation in a solution when it is stuck in a local optimum.

The shaking procedure consists of applying to the current solution p moves chosen among the following: (1) change of execution order of two jobs on the same machine; (2) change of execution order of two jobs belonging to different machines; (3) insertion of a job from a machine into another position of the same machine and (4) insertion of a job from one machine into a position of another machine.

It works as follows: p independent moves are applied consecutively on the current solution s, generating an intermediate solution s'. This solution s' is, then, refined by the VND local search method (line 7 of the Algorithm 1). The level of perturbation p increases after a certain number of attempts to explore the neighborhood without improvement in the current solution. This limit is controlled by the variable *Max*. When p increases, then p random moves (chosen from those mentioned above) are applied to the current solution. Whenever there is an improvement in the current solution, the perturbation returns to its initial level, $p = 2$.

The operation of each type of perturbation is detailed below:

Swap on the Same Machine. This operation consists in randomly choosing two jobs j_1 and j_2 that are, respectively, in the positions x and y of a machine i, and allocate j_1 in the position y and j_2 in the position x of the same machine i.

Swap between Different Machines. This perturbation consists in randomly choosing a job j_1 that is in the position x on a machine i_1 and another job j_2 that is in the position y of the machine i_2. Then, job j_1 is allocated to machine i_2 in position y, and job j_2 is allocated to machine i_1 in position x.

Insertion on the Same Machine. It starts with the random choice of a job j_1 that is initially in the position x of the machine i. Then, a random choice of another position y of the same machine is made. Finally, job j_1 is removed from position x and inserted into position y of machine i.

Insertion between Different Machines. It consists of a random choice of a job j_1 that is in the position x of the machine i_1 and a random choice of position y of the machine i_2. Then, the job j_1 is removed from machine i_1 and inserted into position y of machine i_2.

4.3 Neighborhoods

We used three neighborhood structures to explore the solution space of the problem, and they are described below.

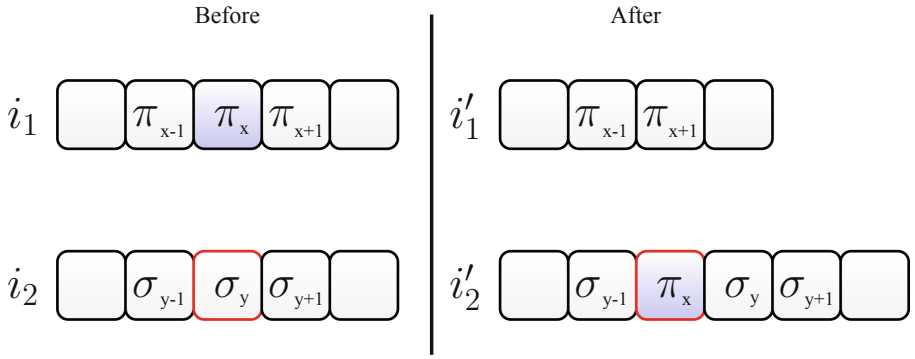

Fig. 1. Insertion move between machines i_1 and i_2.

$\mathcal{N}_1$**: Insertion between Machines.** Let π and σ be two schedules, where $\pi = (\pi_1, \pi_2, \ldots, \pi_t)$ is performed on machine i_1 and $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_r)$ on machine i_2. In these schedules, t and r represent the number of jobs on machines i_1 and i_2, respectively. In this neighborhood, each job $\pi_x \in \pi$ is removed from machine i_1 and added to machine i_2 at position $y \in \{1, \cdots, r\}$. The set of insertion moves of jobs of a machine i_1 in every possible positions of another machine i_2 defines the neighborhood $\mathcal{N}_1(\pi, \sigma)$, which is composed by $t \times (r + 1)$ neighbors.

Figure 1 illustrates an insertion move of a job π_x of a machine i_1 in the position y of the machine i_2. The right side of this figure shows the result of applying this move.

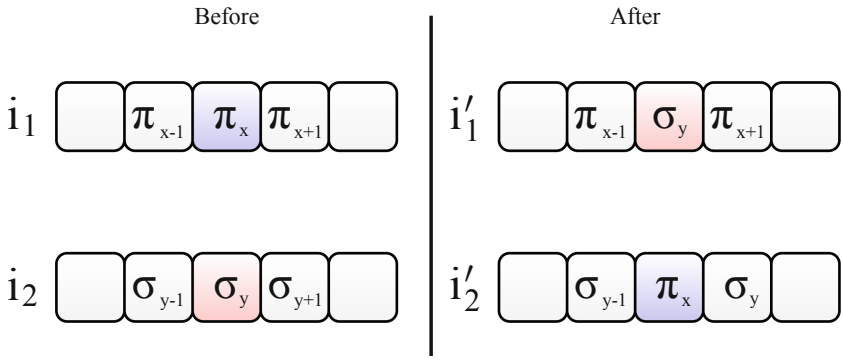

Fig. 2. Swap move between machines i_1 and i_2.

$\mathcal{N}_2$**: Swap Move between Machines.** Let π and σ be two schedules as described above. Let also be two jobs $\pi_x \in \pi$ and $\sigma_y \in \sigma$. The swap move between machines consists in swapping these jobs between these schedules, that is, to move the job π_x to the position y of the machine i_2 and the job σ_y to the position x of the machine i_1. The set of swap moves between machines i_1 and i_2 defines the neighborhood $\mathcal{N}_2(\pi, \sigma)$, formed by $t \times r$ neighbors.
Figure 2 illustrates the swap between two jobs π_x and σ_y, which are initially allocated to machines i_1 and i_2, respectively. After the swap move, the job σ_y is allocated to machine i'_1 and job π_x to machine i'_2.

$\mathcal{N}_3$**: Scheduling by Mathematical Programming.** In this local search, the objective is to determine the best scheduling of the jobs in each machine by applying a MILP formulation. For this, the time-dependent traveling salesman problem (TDTSP) formulation of Bigras et al. [20] was adapted, where the distance between the cities i and j is represented by the sum of the processing time of job i and the setup time between jobs i and j. In addition, a dummy job 0 was added to allow the creation of a Hamiltonian cycle, where 0 represents the first and the last job.
So, the MILP formulation is solved for the sequencing problem in each machine in which the completion time is equal to the makespan. If there is an improvement in the current solution, the local search method returns to the first neighborhood ($\mathcal{N}_1$). If there is no improvement in the current machine and there is another machine whose completion time is equal to the makespan, then the model is applied to this machine. If there is no improvement by applying this formulation, then the exploration in this neighborhood $\mathcal{N}_3$ is ended.
In order to introduce this MILP, the parameters and decision variables are defined and shown in Table 2. The other parameters used by the model are described in Table 1.

Then, the mathematical formulation used as local search strategy in each machine i is given by Eqs. (14)–(18).

Table 2. Parameters and decision variables based on the Bigras et al. [20] model.

Name	Description	Type
N_i	Set of jobs in machine i	Parameter
δ	Any subset of N_i	
Y_{jk}	Equal to 1, if job k is processed directly after job j and equal to 0, otherwise	Decision variable
$C^i_{\max}$	Maximum completion time on machine i	

Objective function:

$$\min C^i_{\max}, \tag{14}$$

Subject to:

$$\sum_{\substack{j \in N_i \cup \{0\}, \\ j \neq k}} Y_{jk} = 1 \qquad \forall k \in N_i \tag{15}$$

$$\sum_{\substack{k \in N_i \cup \{0\}, \\ j \neq k}} Y_{jk} = 1 \qquad \forall j \in N_i \tag{16}$$

$$\sum_{\substack{j \in N_i \cup \{0\} \\ j \neq k}} \sum_{k \in N_i} (S_{ijk} + p_{ik}) Y_{jk} = C^i_{\max} \tag{17}$$

$$\sum_{j \in \delta} \sum_{k \notin \delta} Y_{jk} \geq 1 \qquad \forall \delta \subset N_i, \delta \neq \emptyset \tag{18}$$

$$Y_{jk} \in \{0, 1\} \qquad \forall j \in \{0\} \cup N_i, \ \forall k \in \{0\} \cup N_i, j \neq k \tag{19}$$

$$C^i_{\max} \geq 0 \tag{20}$$

Equation (14) defines the objective function, which is to minimize the completion time of the machine i. Equations (15)–(18) define the constraints for the sub-model. Constraints (15) ensure that every job k has exactly one predecessor job, and the predecessor job of the first job is the dummy job 0. Constraints (3) ensure that each job k has a successor job, and the successor of the last job is the dummy job 0. Constraints (17) compute the completion time on the machine i. Constraints (18) ensure that there is no subcycle, therefore, any subset $\delta \in N_i$ of jobs must have at least one link with another subset complementary to δ, that is, $N_i \backslash \delta$. This strategy is similar to the subtour elimination constraints for the traveling salesman problem, proposed by Bigras et al. [20]. Constraints (19) and (20) define the domain of the decision variables.

The mathematical model has a constraint for each subset of jobs. Thus, in cases where the scheduling problem has many subsets of jobs, the model will

demand a high computational cost. For this reason, the set of constraints (18) was initially disregarded from the model. However, the relaxed model can produce an invalid solution, that is, a solution containing one or more subcycles. If this happens, a new set of constraints for each subcycle is added to the mathematical model to be solved again. In this new set of constraints (18), the set δ is formed by the group of jobs belonging to the subcycle. This process is repeated until a valid solution is found.

For illustrating this situation, consider the matrix below that represents the values of the decision variables for a problem of one machine with five jobs (Table 3).

Table 3. Example of an invalid solution.

Y	*0*	*1*	*2*	*3*	*4*	*5*
0	0	0	0	0	0	**1**
1	0	0	**1**	0	0	0
2	0	**1**	0	0	0	0
3	**1**	0	0	0	0	0
4	0	0	0	**1**	0	0
5	0	0	0	0	**1**	0

Consider that if $Y_{jk} = 1$ then job j immediately precedes job k, and that the first job of the sequence is preceded by the dummy job 0. Then, we have the following subcycles: $\delta_1 = \{5, 4, 3\}$ and $\delta_2 = \{1, 2\}$. This solution is invalid since there should be a single scheduling involving all jobs and not two as can be observed. Figure 3 illustrated this situation:

Thus, a new constraint must be added for any solution that has a subcycle, since this situation does not obey Eq. (18).

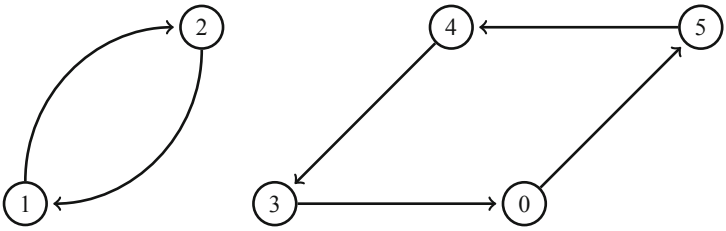

Fig. 3. Representation of an invalid solution.

4.4 Local Search

The local search of our algorithm is done by a VND procedure, that uses the three neighborhood structures $\mathcal{N}_1$, $\mathcal{N}_2$ and $\mathcal{N}_3$ defined in Sect. 4.3. Its pseudo-code is presented in Algorithm 3.

Algorithm 3. VND.

```
   input : s, N
 1 k ← 1;
 2 while (k ≤ 3) do
 3 |   s'' ← BestNeighbor(s, N_k);
 4 |   if (f(s'') < f(s)) then
 5 |   |   s ← s'';
 6 |   |   k ← 1;
 7 |   end
 8 |   else
 9 |   |   k ← k + 1 ;
10 |   end
11 end
12 return s ;
```

Thus, the VND returns a local optimum in relation to all three neighborhoods $\mathcal{N}_1$, $\mathcal{N}_2$ and $\mathcal{N}_3$.

5 Computational Experiments

The Smart GVNS algorithm was coded in C++ language and the tests were performed on a microcomputer with the following configurations: Intel (R) Core (TM) i7 processor with clock frequency 2.4 GHz, 8 GB of RAM and with a 64-bit Ubuntu operating system installed. The mathematical heuristic, used as local search, was implemented using the Gurobi API [21] for the C++ language.

The proposed algorithm was tested in three sets of instances available by Rabadi et al. [18]: Balanced, Process Domain, and Setup Domain. Each set is formed by 18 groups of instances, and each group contains 15 instances, totaling 810 instances. In the first set, the processing time and the setup time are balanced. In the second, the processing time is dominant in relation to the setup time and in the third, the setup time is dominant in relation to the processing time.

5.1 Parameter Tuning

The implementation of the e-SGVNS algorithm requires the calibration of two parameters: *MaxP* and *MaxSameLevelP*, which are defined in Algorithm 1.

The Irace package [22] was used to tune the values of these parameters. Irace is an algorithm implemented in R that implements an iterative procedure having as main objective to find the most appropriate configurations for an optimization algorithm, considering a set of instances of the problem.

We tested the following values for the two parameters of the e-SGVNS: $MaxP$ and $MaxSameLevelP \in \{3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$. The best configurations returned by Irace were $MaxP = 5$ and $MaxSameLevelP = 12$.

5.2 Stopping Criterion

As stopping criterion of the e-SGVNS algorithm and for a fair comparison, the average execution time of ACOII by Arnaout et al. [10] was used. Their time was divided by 2.0 because our computer is approximately 2.0 times faster[1] than the computer used in [10] according to PassMark [23]. Cota et al. [12] also used the same stopping criterion to report the results of LA-ALNS algorithms (Table 4).

Table 4. Time limit for e-SGVNS Algorithm in minutes (the same of SGVNS).

Jobs	Machines		
	8	10	12
80	2.78	2.78	2.86
100	4.97	5.13	5.13
120	6.88	7.09	7.09

The WO algorithm used a time limit different from others. Thus, it is compared only with respect to its results and not to its efficiency.

6 Results

Tables 5, 6 and 7 compare the results of the proposed e-SGVNS algorithm with those of ACOII reported by Arnout et al. [10], LA-ALNS reported by Cota et al. [12], WO described by Arnaout [9] and SGVNS by Rego and Souza [6] in relation to the average Relative Percent Deviation (RPD) in each group of 15 instances. For each instance l, the RPD is calculated by:

$$RPD_l = \frac{f_l^{alg} - f_l^{\star}}{f_l^{\star}} \times 100 \tag{21}$$

where f_l^{alg} is the value of the objective function for the algorithm alg in relation to the instance l, while $f_l^{\star}$ represents the Lower Bound (LB) for the l-th instance reported by Al-Salem [24].

[1] https://www.cpubenchmark.net/compare/Intel-i7-870-vs-Intel-Pentium-4-3.00GHz/832vs1074.

Table 5. Average RPD in Balanced instances.

m	n	ACOII	LA-ALNS	SGVNS	WO	e-SGVNS
2	80	1.24	1.8	1.49	1.41	**1.21**
	100	1.08	1.51	1.33	1.35	**0.94**
	120	0.92	1.36	1.16	1.06	**0.79**
4	80	3.97	3.54	4.53	3.83	**3.29**
	100	3.54	2.96	4.22	3.4	**2.91**
	120	3	2.75	3.73	3.1	**2.41**
6	80	7.09	**5.64**	7.27	5.89	5.88
	100	5.58	**4.25**	6.27	4.86	4.57
	120	4.52	**3.73**	5.84	4.52	3.87
8	80	7.41	**5.59**	8.04	6.0	6.41
	100	8.11	**5.85**	8.62	6.67	7.0
	120	5.7	**4.35**	6.74	5.24	4.9
10	80	8.14	**6.79**	9.26	6.96	7.72
	100	8.15	**5.54**	8.38	6.35	6.58
	120	6.99	**4.01**	8.05	6.13	5.88
12	80	11.97	–	13.82	**10.39**	11.07
	100	12.18	–	13.71	**11.18**	11.47
	120	7.6	–	9.03	**6.82**	7.02

Table 6. Average RPD in the Process Domain instances.

m	n	ACOII	LA-ALNS	SGVNS	WO	e-SGVNS
2	80	0.80	1.14	0.91	0.90	**0.71**
	100	1.65	0.95	0.80	1.16	**0.63**
	120	1.49	0.83	0.66	1.01	**0.46**
4	80	2.50	2.16	2.79	2.25	**2.02**
	100	2.07	**1.70**	2.26	1.98	1.74
	120	2.14	1.88	2.24	1.98	**1.56**
6	80	5.48	5.33	6.07	**5.24**	5.33
	100	4.07	**3.00**	4.35	3.35	3.24
	120	2.97	2.36	3.09	2.52	**2.32**
8	80	4.44	–	4.81	**3.50**	3.66
	100	6.52	–	6.71	5.40	**5.29**
	120	3.69	–	3.92	**2.91**	2.97
10	80	4.44	3.79	5.48	**3.70**	4.27
	100	4.91	**3.26**	4.97	3.56	3.93
	120	4.55	**3.19**	4.52	3.47	3.56
12	80	9.07	–	10.16	**7.92**	8.57
	100	10.36	–	11.09	**9.50**	9.86
	120	4.64	–	5.05	**3.71**	4.21

Table 7. Average RPD in the Setup Domain instances.

m	n	ACOII	LA-ALNS	SGVNS	WO	e-SGVNS
2	80	0.77	1.06	0.83	0.87	**0.66**
	100	1.43	0.90	0.74	1.07	**0.59**
	120	1.37	0.85	0.70	1.04	**0.50**
4	80	2.49	1.96	2.64	2.22	**1.86**
	100	2.16	1.84	2.36	2.04	**1.67**
	120	1.90	1.61	2.09	1.82	**1.50**
6	80	5.64	5.18	6.00	**5.01**	5.39
	100	4.12	**2.99**	4.73	3.31	3.22
	120	2.82	2.41	3.35	2.52	**2.35**
8	80	4.74	**3.30**	4.71	3.49	3.53
	100	6.54	**5.05**	6.72	5.32	5.43
	120	3.78	**2.59**	4.11	2.96	2.88
10	80	4.67	**3.90**	5.76	4.00	4.54
	100	4.77	**3.17**	4.98	3.43	3.66
	120	4.28	**3.22**	4.60	3.58	3.42
12	80	8.84	–	10.24	**7.85**	8.73
	100	9.95	–	16.42	**9.81**	9.91
	120	4.25	–	5.21	**3.58**	3.90

In these tables, the first and second columns represent the number of machines and jobs, respectively. In the subsequent columns are the average RPD for ACOII, LA-ALNS, SGVNS, WO and e-SGVNS algorithms, respectively.

According to Tables 5, 6 and 7, the LA-ALNS algorithm was superior in 20 groups of instances, while the WO algorithm was superior in 14 groups of instances and the e-SGVNS algorithm was superior in 20 groups of instances. The results from SGVNS and ACOII algorithms were outperformed in all instance sets. Considering the presented results, it is possible to affirm that the LA-ALNS algorithm obtained the best average results, even though it was not applied to all the instances made available in [18].

The proposed algorithm presented a value for RPD less than 0 in instances with two machines. If we consider instances with 4 machines, the RPD was always less than 2, while for instances with up to 8 machines, the RPD was always less than 3. For the other instances, the RPD was always less than 4. These results indicate that the proposed method obtained a better performance in instances with fewer machines, in which the solution space is smaller. In other cases, the method has lower performance, given the high computational cost of the mathematical heuristic, which is used as one of the local search operators.

6.1 Statistical Analysis

A hypothesis test was performed to verify if the differences between the results presented by the algorithms are statistically significant. Therefore, the following hypothesis test was used:

$$\begin{cases} H_0 : \mu_1 = \mu_2 = \mu_3 = \mu_4 = \mu_5 \\ H_1 : \exists i, j \mid \mu_i \neq \mu_j \end{cases}$$

in which μ_1, μ_2 and μ_3 are the average RPDs for ACOII, LA-ALNS, SGVNS, WO and e-SGVNS, respectively.

An exploratory analysis of the data was performed in order to better understand the data of the samples before the application of the statistical test.

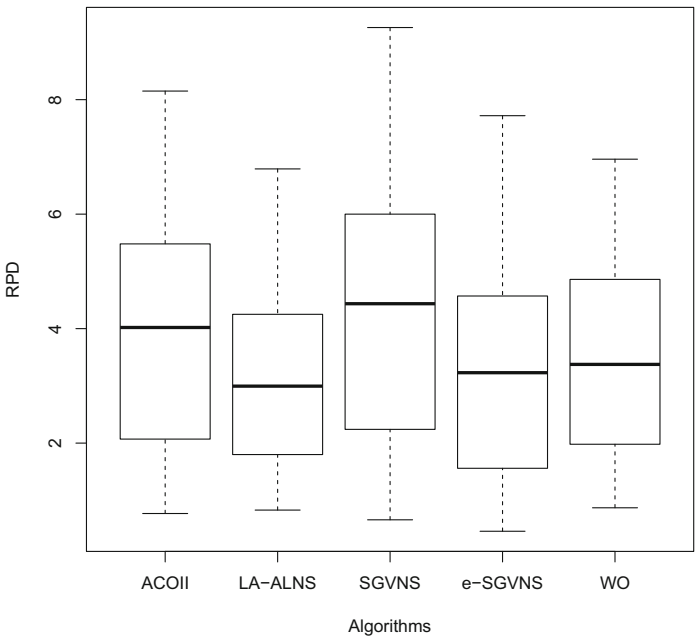

Fig. 4. Boxplot of the results.

Figure 4 shows the boxplot plot containing the sample distribution of the RPD values for the collected samples:

Before performing the hypothesis test, it is necessary to decide between test types, parametric or non-parametric. Generally, parametric tests are more powerful; however, they require three assumptions:

1. Normality: every sample must originate from a population with normal distribution,

2. Independence: the samples shall be independent of each other,
3. Homoscedasticity: every sample must have a population of constant variance.

The Shapiro-Wilk normality test was applied to the samples and its results are shown in Table 8:

Table 8. Shapiro-Wilk normality test.

Algorithm	p-value
ACO II	0.04632
LA-ALNS	0.06239
SGVNS	0.09066
e-SGVNS	0.0737
WO	0.0365

Considering a significance level of 0.05, the results presented above indicate that the samples of the ACOII and WO algorithms come from populations with normal distribution, since the p-values presented are lower than the level of significance. However, the test does not present evidence that the LA-ALNS, SGVNS and e-SGVNS algorithm samples come from a normal population.

Therefore, it was decided to use the Pairwise Wilcoxon test, which calculates pairwise comparisons between group levels with corrections for multiple testing.

The Pairwise Wilcoxon test for the samples of the average results of the ICOII, LA-ALNS, SGVNS, WO and e-SGVNS algorithms are presented in Table 9. In this comparison, we excluded instance sets in which Cota et al. [12] did not report the results of LA-ALNS algorithm.

Table 9. Pairwise comparisons using Wilcoxon test in all algorithms.

	ACOII	e-SGVNS	LA-ALNS	SGVNS
e-SGVNS	1.70×10^{-7}	–	–	–
LA-ALNS	6.70×10^{-6}	1	–	–
SGVNS	8.6×10^{-4}	1.70×10^{-7}	7.70×10^{-8}	–
WO	5.90×10^{-6}	0.2819	2.4×10^{-4}	5.70×10^{-6}

According to Table 9, the observed differences are statistically significant for all algorithm pairs, except to (e-SGNVS, LA-ALNS) and (e-SGNVS, WO).

Table 10 displays the Pairwise Wilcoxon test considering the average RPD of algorithms that were tested in all instances.

Considering that this p-value is much lower than 0.05, then the null hypothesis of equality between the means is rejected and it is concluded that there is evidence that at least two populations have different distribution functions.

Table 10. Pairwise comparisons using Wilcoxon test in all instances.

	ACOII	e-SGVNS	SGVNS
e-SGVNS	1.00×10^{-9}	–	–
SGVNS	2.60×10^{-6}	1.00×10^{-9}	–
WO	2.10×10^{-8}	1	1.60×10^{-8}

As can be seen in Table 9, there is a statistically significant difference between the e-SGVNS algorithm and the SGVNS, ACOII algorithms.

7 Conclusions

This work dealt with the unrelated parallel machine scheduling problem with sequence-dependent setup times, aiming to minimize the makespan.

Since it is NP-hard, a hybrid heuristic algorithm was developed. The proposed algorithm, named Enhanced Smart General Variable Neighborhood Search (e-SGVNS), combines heuristic and exact optimization strategies to explore the solution space of the problem. The exact strategy works as local search and consists of applying a mathematical programming formulation based on the time-dependent traveling salesman problem to get the optimal solution to the sequencing problem on each machine. In turn, the heuristic strategy, in turn, explores neighborhoods based on swap and insertion moves.

The e-SGVNS was tested in benchmark instances from literature and its results were compared to four other literature methods (ACOII, LA-ALNS, SGVNS and WO).

The statistical analysis of the average results produced by the algorithms proved that e-SGVNS is statistically better than the SGVNS and ACOII algorithms. On the other hand, there is no statistical evidence of significant difference among the average results of the e-SGVNS, LA-ALNS and WO algorithms.

Overall, the e-SGVNS algorithm performed best on small instances, with up to 4 machines and up to 120 jobs, regardless of instance type.

As future work, we intend to test other mathematical programming formulations to perform the exact local search as, for instance, to apply a mixed integer linear programming formulation that considers two machines instead of a single one.

Acknowledgments. The authors gratefully thank *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES) - Finance Code 001, *Fundação de Amparo à Pesquisa do Estado de Minas Gerais* (FAPEMIG, grant PPM/CEX/FAPEMIG/676-17), *Conselho Nacional de Desenvolvimento Científico e Tecnológico* (CNPq, grant 307915/2016-6), *Universidade Federal de Ouro Preto* (UFOP) and *Universidade Federal dos Vales do Jequitinhonha e Mucuri* (UFVJM) for supporting this research. The authors also thank the anonymous reviewers for their valuable comments.

References

1. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W., Bohlinger, J.D. (eds.) Complexity of Computer Computations. The IBM Research Symposia Series, vol. 40, pp. 85–103. Springer, Boston (1972). https://doi.org/10.1007/978-1-4684-2001-2_9
2. Lopes, M.J.P., de Carvalho, J.V.: A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times. Eur. J. Oper. Res. **176**(3), 1508–1527 (2007)
3. Avalos-Rosales, O., Alvarez, A.M., Angel-Bello, F.: A reformulation for the problem of scheduling unrelated parallel machines with sequence and machine dependent setup times. In: Proceedings of the 23rd International Conference on Automated Planning and Scheduling - ICAPS, Rome, Italy, pp. 278–283 (2013)
4. Lee, Y.H., Pinedo, M.: Scheduling jobs on parallel machines with sequence-dependent setup times. Eur. J. Oper. Res. **100**(3), 464–474 (1997)
5. Tran, T.T., Araujo, A., Beck, J.C.: Decomposition methods for the parallel machine scheduling problem with setups. INFORMS J. Comput. **28**(1), 83–95 (2016)
6. Rego., M.F., Souza, M.: Smart general variable neighborhood search with local search based on mathematical programming for solving the unrelated parallel machine scheduling problem. In: Proceedings of the 21st International Conference on Enterprise Information Systems - Volume 1: ICEIS, INSTICC, pp. 287–295. SciTePress (2019)
7. Mladenović, N., Dražić, M., Kovačevic-Vujčić, V., Čangalović, M.: General variable neighborhood search for the continuous optimization. Eur. J. Oper. Res. **191**(3), 753–770 (2008)
8. Santos, H.G., Toffolo, T.A., Silva, C.L., Vanden Berghe, G.: Analysis of stochastic local search methods for the unrelated parallel machine scheduling problem. Int. Trans. Oper. Res. **26**(2), 707–724 (2019)
9. Arnaout, J.P.: A worm optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. Ann. Oper. Res. **285**, 273–293 (2019)
10. Arnaout, J.P., Musa, R., Rabadi, G.: A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines - part II: enhancements and experimentations. J. Intell. Manuf. **25**(1), 43–53 (2014)
11. Arnaout, J.P., Rabadi, G., Musa, R.: A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. J. Intell. Manuf. **21**(6), 693–701 (2010)
12. Cota, L.P., Guimarães, F.G., de Oliveira, F.B., Souza, M.J.F.: An adaptive large neighborhood search with learning automata for the unrelated parallel machine scheduling problem. In: 2017 IEEE Congress on Evolutionary Computation (CEC), pp. 185–192. IEEE (2017)
13. Fanjul-Peyro, L., Ruiz, R.: Iterated greedy local search methods for unrelated parallel machine scheduling. Eur. J. Oper. Res. **207**(1), 55–69 (2010)
14. Mokotoff, E., Jimeno, J.: Heuristics based on partial enumeration for the unrelated parallel processor scheduling problem. Ann. Oper. Res. **117**(1), 133–150 (2002)
15. Ghirardi, M., Potts, C.N.: Makespan minimization for scheduling unrelated parallel machines: a recovering beam search approach. Eur. J. Oper. Res. **165**(2), 457–467 (2005)
16. Vallada, E., Ruiz, R.: A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. Eur. J. Oper. Res. **211**(3), 612–622 (2011)

17. Kurz, M., Askin, R.: Heuristic scheduling of parallel machines with sequence-dependent set-up times. Int. J. Prod. Res. **39**(16), 3747–3769 (2001)
18. Rabadi, G., Moraga, R.J., Al-Salem, A.: Heuristics for the unrelated parallel machine scheduling problem with setup times. J. Intell. Manuf. **17**(1), 85–97 (2006)
19. Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**(11), 1097–1100 (1997)
20. Bigras, L.P., Gamache, M., Savard, G.: The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. Discrete Optim. **5**(4), 685–699 (2008)
21. LLC Gurobi Optimization: Gurobi optimizer reference manual (2018). http://www.gurobi.com
22. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., Birattari, M.: The irace package: iterated racing for automatic algorithm configuration. Oper. Res. Perspect. **3**, 43–58 (2016)
23. PassMark: CPU benchmarks (2018). https://www.cpubenchmark.net/. Accessed 12 Feb 2018
24. Helal, M., Rabadi, G., Al-Salem, A.: A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times. Int. J. Oper. Res. **3**(3), 182–192 (2006)