An Adaptive Large Neighborhood Search with Learning Automata for the Unrelated Parallel Machine Scheduling Problem

Luciano Perdigão Cota*, Frederico Gadelha Guimarães†, Fernando B. de Oliveira‡, Marcone J. Freitas Souza§
*Graduate Program in Electrical Engineering, Federal University of Minas Gerais, Belo Horizonte, Brazil
†Department of Electrical Engineering, Universidade Federal de Minas Gerais, Belo Horizonte, MG, Brazil
‡Department of Computer and Systems, Universidade Federal de Ouro Preto, João Monlevade, Brazil
§Department of Computer Science, Universidade Federal de Ouro Preto, Ouro Preto, Brazil
Email: {lucianocota, fredericoguimaraes}@ufmg.br, fernando@decea.ufop.br, marcone@iceb.ufop.br

***Abstract*—This work deals with the Unrelated Parallel Machine Scheduling Problem with Setup Times, with the objective of minimizing the makespan. It is proposed an Adaptive Large Neighborhood Search (ALNS) metaheuristic using Learning Automata (LA) to adapt the probabilities of using removal and insertion heuristics and methods. A computable function in the LA updates the probability vector for selecting the actions, corresponding to six removal and six insertion methods. We also propose a new insertion method based on Hungarian algorithm, which is applied to solve subproblems optimally. Computational experiments are performed to verify the performance of the proposed method. A set of instances available in the literature with problems up to 150 jobs and 10 machines is employed in the experiments. The proposed LA-ALNS is compared against three other algorithms from the literature. The results suggest that our algorithm has better performance in most of cases (88%) under the defined conditions of experiments. Statistical tests also suggest that LA-ALNS is better than the other algorithms from the literature. The proposed method is able to automatically choose the most suitable heuristics for the instance of the problem, through adaptation and learning in the Learning Automata.**

I. Introduction

Parallel machine scheduling problems with sequence-dependent setup times are relevant to many manufacturing and industrial processes, attracting a special interest of several researchers [1]. Many real world problems can be approached with a parallel machine scheduling model, since in a more abstract sense, it deals with the efficient allocation of resources, generally called machines, to different tasks, usually called jobs, optimizing a given cost function. Many problems in airport management and hospital allocation of resources can be viewed as a parallel machine scheduling problem. The efficient allocation of resources to jobs in these applications can result in significant cost reduction and productivity increase. Figure 1 illustrates a scenario of machine scheduling in an industrial context. In that scenario, two machines are making different products. The aim is related to minimizing the makespan, which in turn results in the maximization of production.

In this paper we focus on the *Unrelated Parallel Machine Scheduling Problem with Setup Times* (UPMSP-ST). UPMSP-ST is relevant to several industrial processes and segments, such as textile, chemical, semiconductor and paper

978-1-5090-4601-0/17/$31.00

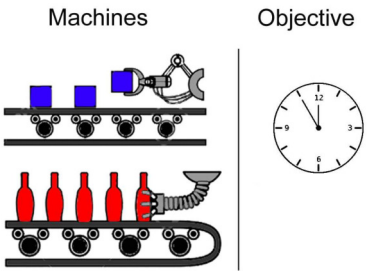


Fig. 1: An example of machine scheduling in an industrial process.

[2], [3]. This problem is also studied in the literature due to its theoretical relevance. It belongs to the $\mathcal{NP}$-hard class [4] and is a generalization of the *Parallel Machine Scheduling Problem with Identical Machines and without Setup Times*. The proof is presented in [5], [6].

In UPMSP-ST, there is a set of jobs $N = \{1, ..., n\}$ and a set of machines $M = \{1, ..., m\}$, with the following features: i) each job $j \in N$ is allocated to only one machine $i \in M$; ii) the time to process $j \in N$ on a machine $i \in M$ is defined by p_{ij} (processing time); iii) there is a setup time S_{ijk} to process job $k \in N$ after job $j \in N$ on machine $i \in M$, in that order. The objective is to allocate all N jobs on M machines and minimizing the *makespan*. Based on this description, the UPMPS-ST is defined as $R_M|S_{ijk}|C_{\max}$ [7]. R_M indicates unrelated machines, S_{ijk} indicates that the setup times are sequence-dependent and $C_{\max}$ defines the objective function as the *makespan*.

An example with six jobs and two machines is presented as follows. The processing times on machines $M1$ and $M2$ are given below. Setup times of those machines are shown in Table I.

$$M1 \ : \ p_{1j} = \{1, 87, 28, 32, 38, 9\}$$

$$M2 \ : \ p_{2j} = \{4, 21, 68, 17, 43, 48\}$$

Figure 2 illustrates a solution for this example. Shading lines represent the setup times. Jobs 3, 5 and 1 are allocated in machine $M1$ and jobs 4, 2 and 6 are allocated in machine $M2$, in this order.

TABLE I: Setup times of machines M_1 and M_2

M_1	1	2	3	4	5	6	M_2	1	2	3	4	5	6
1	3	1	8	1	3	9	1	8	5	1	6	1	7
2	4	3	7	3	7	8	2	6	1	7	7	6	2
3	7	3	1	2	3	5	3	7	6	3	9	6	9
4	3	8	3	2	5	2	4	3	7	3	2	1	7
5	8	3	7	9	1	5	5	5	8	5	6	1	9
6	8	8	1	2	2	3	6	7	4	1	7	9	2

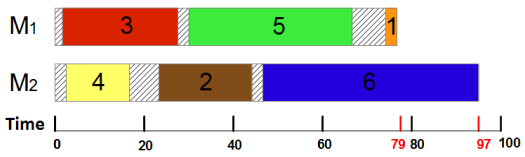


Fig. 2: A solution for the example problem

The cost of machine $M1$ is given by $C_{M1} = S_{103} + p_{13} + S_{135} + p_{15} + S_{151} + p_{11} = 79$, and the cost of machine $M2$ is given by $C_{M2} = S_{204} + p_{24} + S_{242} + p_{22} + S_{226} + p_{26} = 97$. The *makespan* of the solution is given by the completion time for the machine that finishes last. In that case, that machine is $M2$ and the makespan is thus 97.

Mathematical models are not able to solve large instances of the UPMPS-ST. In a recent work presented by [8], mathematical models are developed to solve instances up to 60 jobs optimally in two to three hours. Therefore, heuristic methods are still a practical approach to deal with instances with more than 60 jobs efficiently and in a few minutes. In this paper we propose an Adaptive Large Neighborhood Search (ALNS) [9] using Learning Automata for adjusting the probabilities of removal and insertion heuristics that are essential in ALNS. The algorithm operates as follows: i) an initial solution is created by means of a semi-greedy heuristic inspired on the constructive phase of the Greedy Randomized Adaptive Search Procedures (GRASP) metaheuristic [10]. ii) ALNS iterations are performed until the stopping criterion is met. On each iteration, a method to remove jobs and another one to insert jobs into the solution are applied to explore the search space in an efficient manner. A roulette method is used to select those methods according to their probabilities. Among the heuristics used, we include a new insertion heuristic based on the Hungarian method, which is able to solve quadratic assignment problems in polynomial time. iii) After removal and insertion operation, the Random Variable Neighborhood Descent (RVND) is used as a local search [11]. Local searches are operated randomly in RVND, which is different from Variable Neighborhood Descent (VND) [12]. iv) In the end of each iteration, the performance of removal and insertion processes is evaluated and used by the Learning Automata (LA) to adapt probabilities.

A set of instances available in [13] is used herein to evaluate the performance of the proposed LA-ALNS. The results are compared with the metaheuristic AIRP [14], and the ant colony based methods ACO [15] and ACOII [16] (an updated version of ACO). The ACOII proposed in [16] has been compared with a number of different metaheuristics including genetic algorithms, providing better results than other methods in the literature for the instances in [13].

The rest of this paper is organized as follows. Section II presents a literature review of UPMPS-ST. Learning Automata and Hungarian method are found in Section III. The proposed algorithm is presented in Section IV. Section V reports the computational experiments and results. Future avenues for research are given in the conclusion in Section VI.

II. Literature review

There are in the literature some works about similar problems to the UPMSP-ST. A few of those ones are described here. Genetic Algorithm (GA), Simulated Annealing (SA) and Tabu Search (TS) were applied in [17] to solve a problem without setup times. A TS method was also proposed in [18] to solve that same problem. A problem with dependent setup time for jobs only, and minimizing the weighted makespan as objective was studied in [19]. In that work, seven constructive heuristics were proposed. SA was used by [20], [21] to deal with a problem with dependent setup time for jobs only, and minimizing the total delay. Six different versions of a TS were proposed by [22] to minimize the total delay in a problem with dependent setup time for jobs only.

With regard to the UPMSP-ST, some works are presented as follows. A partitioning heuristic was defined in [23], which combines a constructive heuristic, a local search, and a heuristic inspired by the asymmetric traveling salesman problem. A heuristic to Randomized Priority Search (Meta-RaPS), a mathematical model and a set of instances were developed by [2]. Those instances are available in [13]. An Ant Colony Optimization (ACO) algorithm was presented in [15]. A Restricted Simulated Annealing (RSA) was defined by [24], and is based on eliminating inefficient job moves. Two GAs were proposed by [25], and a mathematical model quite similar to [2] was also defined. Besides, a new set of instances were made available in [26]. A Bee Colony optimization algorithm was presented in [27]. The mathematical model defined in [25] was improved by [28], [8]. The change allowed to solve instances up to 60 jobs optimally in two to three hours in average. An improvement of ACO presented in [15] was proposed by [16]. That new algorithm is called ACOII, and performs better than a set of other algorithms in the literature for those instances available in [13]. In the works of [29], [14], [30], [31], different algorithms were proposed based on the combination of Iterated Local Search (ILS) [32] and VND metaheuristics. The algorithm defined in [14], namely AIRP, has shown better performance for those instances available in [26].

The goal of this work is to develop an ALNS and incorporate Learning Automata to adapt the probabilities of removal and insertion heuristics. One of the main insertion methods proposed here is based on the Hungarian algorithm, which is applied to solve subproblems optimally. As far as we know, Learning Automata has never been applied to the adaptive phase of ALNS. Moreover, the use of the Hungarian

algorithm as an heuristic in the UPMSP-ST is also one novelty of this work. We intend to propose an algorithm that is able to automatically choose the most suitable heuristics for the instance of the problem, through adaptation and learning in Learning Automata. Besides, this work motivates new research about ALNS, Learning Automata and Hungarian algorithm, and their application to the machine scheduling problem with setup times.

III. Background

This section describes the Hungarian algorithm and the Learning Automata process.

A. Hungarian algorithm

The Hungarian algorithm has been initially defined by [33] [34]. This algorithm attempts to solve assignment problems, which are known in graph theory as an optimal correlation for a bipartite graph. The Hungarian algorithm is based on König and Egerváry ideas [35]. König theorem presents that in a bipartite graph, the maximum cardinality of a correlation is also the minimum number of nodes that cover all edges.

Given a bipartite graph with a weight function w, the cost of optimal correlation of K is defined as $\gamma(K) = \sum_e^K \gamma(e)$, in which the cost $\gamma(e)$ of edge e is represented by $\gamma(e) = C - w(e)$, wherein C is the maximum weight of all edges. Thus, the optimal correlation in a bipartite graph with regard to the weight function is equivalent to solving the assignment problem for a matrix M, in which $m_{ij} = C - w_{ij}$, where w_{ij} is the weight of the edges between nodes i and j. This corresponds to finding an optimal correlation with minimal cost.

The method requires a square matrix $n \times n$ and a conversion from the maximum assignment problem to the minimum one is initially done by replacing each w_{ij} by $C - w_{ij}$. The method consists of five steps, given as follows.

1) Subtract the smallest input from each line from all inputs on the same line. Thereby, each line will have at least one zero entry and all other entries are nonnegative.
2) Subtract the smallest input from each column from all inputs on the same column. Thereby, each column will have at least one zero entry and all other entries are nonnegative.
3) Make a dash along rows and columns such that all zero entries of the cost-matrix are crossed-out. To do this, use a minimum number of dashes.
4) Optimization test
   - Let n be the total number of rows or columns of the cost-matrix. If the minimum number of dashes needed to cover the zeros is n, then an optimal allocation of zeros is possible and we close the procedure.
   - If the minimum number of dashes needed to cover the zeros is less than n, then an optimal zeros allocation is not yet possible. In this case, go to step 5.
5) Determine the smallest unmarked entry. Subtract this value from all unmarked entries and add it to all inputs with dash. Return to step 3.

The solution of assignment problems by means of brute-force search has factorial time complexity. Meanwhile, the Hungarian algorithm might solve those problems in polynomial time, more precisely $\theta(n^3)$.

B. Learning Automata

A Learning Automata is an adaptive decision unit that improves its performance by learning to choose a better action from a finite set of allowed actions by means of repeated interactions in a random environment [36]. The action is randomly chosen with a probability distribution in a set of actions. At each interaction, the selected action is used as input to the random environment for further learning.

Basically, an LA is defined as $(\phi, \alpha, \beta, A, \pi, p)$ [37], where ϕ is a set of internal states; α is a set of outputs or actions of learning automata; β is a set of responses from the environment; A is a Learning Automaton; $\pi : \phi \mapsto \alpha$ is a function that maps the current state into a current output; p is a vector that defines a selection probability of an action on each stage. Figure 3 illustrates the relation between Learning Automata and its random environment.

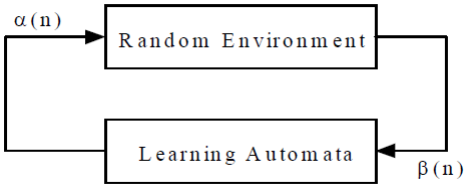


Fig. 3: Relation between Learning Automata and its random environment [38].

When an action is applied to the random environment, it responds back to the LA with a noise signal. LA uses that response to adjust the probabilities of internal action through its learning algorithm. Let β be a response from the environment in step k with $\beta \in \{0, 1\}$, in which responses 0 and 1 mean "agreeable" and "disagreeable", respectively. If the response is "agreeable", the probability of internal action might be updated with Equation (1). Otherwise, if the response is "disagreeable", the probability is updated with Equation (2) [39].

$$p_j(k+1) = \begin{cases} p_j(k) + a(1 - p_j(k)) & \text{if } i = j \\ p_j(k)(1-a) & \text{if } i \neq j \end{cases} \quad (1)$$

$$p_j(k+1) = \begin{cases} p_j(k)(1-b) & \text{if } i = j \\ \frac{b}{r-1} + p_j(k)(1-b) & \text{if } i \neq j \end{cases} \quad (2)$$

For Eq. (1) and (2), a and b are named reward and penalty parameters, respectively. The number of actions that might be chosen by the automaton is defined by $r = |\alpha|$. When $a = b$, the learning algorithm is named linear reward-penalty

(L_{R-P}). When $a << b$, it is named reward-ϵ linear penalty ($L_{R\epsilon P}$), and when b is equal to zero, the learning algorithm is named linear reward-inaction L_{R-I} [39].

IV. The proposed LA-ALNS algorithm

A. Representation and solution evaluation

A solution is represented as an array of integers with m positions, in which m is the number of machines. On each position of array, a list is associated and defines the allocated jobs for each machine. Figure 4 represents a solution for an instance with six jobs and two machines. Jobs 3, 5 and 1 are allocated to machine 1 in that order. Jobs 4, 2 and 6 are allocated to machine 2, in that order.

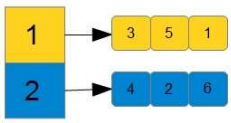


Fig. 4: Representation for a solution.

A solution is evaluated by its makespan, which is the completion time of the machine that finishes last.

B. ALNS algorithm

The proposed algorithm to solve the UPMPS-ST is based on the Adaptive Large Neighborhood Search metaheuristic [9], using Learning Automata for the adaptive phases of removal and insertion methods. A^- is the automaton defined for the removal process and A^+ for the insertion one. The removal and insertion heuristics represent the set of actions for each LA, α^- and α^+, respectively. An initial solution is generated by means of a semi-greedy heuristic based on constructive phase of GRASP. Local search is performed by applying RVND. The pseudocode is shown in Algorithm 1.

The proposed LA-ALNS performs as follows. At first, an initial solution is created using semi-greedy heuristic with time limit of 1% of the total execution time. The initial temperature is set such that there is 50% of chance of acceptance of a solution that is 10% worse than the initial solution. The probabilities of removal and insertion methods in the LA are updated every K iterations, according to the parameter K. This parameter was defined by $6 \times \max(|\alpha^-|, |\alpha^+|)$, obtained previously by empirical tests.

These steps are performed on each iteration:

1) A removal method $\alpha_i^- \in \alpha^-$ and an insertion method $\alpha_j^+ \in \alpha^+$ are selected by means of a roulette method according to their probabilities;
2) q jobs are removed from the current solution (s') using α_i^- method and then inserted using α_j^+ method. The parameter q was defined as 5% of the total number of jobs using empirical tests. The pair of a removal and insertion heuristic acts as a biased perturbation applied to the current solution;
3) The $RVND$ method is applied on solution s' as local search;
4) After that, the probabilities p^- and p^+ are updated using Equation (1), if the solution s' is accepted, or Equation (2) if the solution is not accepted.
5) At every K iterations, the probabilities of removal and insertion methods are updated into the LA. In other words, learning occurs every iteration but only after K iterations the probability values are updated into the LA. This is done to give time to the LA to experiment some actions in the environment before updating the selection probabilities;
6) At the end of LA-ALNS, the best solution found is returned.

A solution s' might be accepted in three different situations: i) if it is better than the best solution; ii) if it is

Algorithm 1: Proposed ALNS

input : $t_{\max}, K, q, a_1, a_2, a_3, b_1$
output: s_{best}

Define LA $(\phi^-, \alpha^-, \beta, A^-, \pi^-, p^-)$;
Define LA $(\phi^+, \alpha^+, \beta, A^+, \pi^+, p^+)$;
$s \leftarrow$ `constructive procedure` ();
$s_{best} \leftarrow s$;
$T \leftarrow (0.1 \times f(s))/\ln 2$;
while $currentTime \leq t_{\max}$ **do**
  Select α_i^- using roulette method;
  Select α_j^+ using roulette method;
  Remove q jobs from s using α_i^-;
  Insert the removed jobs on s using α_j^+;
  $s' \leftarrow$ `RVND` (s) ;
  if $f(s') < f(s_{best})$ **then**
    $s_{best} \leftarrow s'$; $s \leftarrow s'$;
    $a = a_1$; $b = 0$;
    Update p^- and p^+ using Eq. (1)-(2);
  end
  else if $f(s') < f(s)$ **then**
    $s \leftarrow s'$;
    $a = a_2$; $b = 0$;
    Update p^- and p^+ using Eq. (1)-(2);
  end
  else if `random` $< e^{-\frac{(f(s')-f(s))}{T}}$ **then**
    $s \leftarrow s'$;
    $a = a_3$; $b = 0$;
    Update p^- and p^+ using Eq. (1)-(2);
  end
  else
    $a = 0$; $b = b_1$;
    Update p^- and p^+ using Eq. (1)-(2);
  end
  if $k \mod K = 0$ **then**
    `updateLA`(A^-, A^+, p^-, p^+);
  end
  $T \leftarrow 0.99 \times T$;
end
return s_{best}

better than the current solution; iii) if it is worse than the current solution, with probability given by the temperature parameter. For each case of solution acceptance, a different reward parameter (a) is applied to update LA. Otherwise, if a solution is not accepted, a penalty parameter (b) is used in updating process. Those parameter values were defined as: i) $a_1 = 0.2$; ii) $a_2 = 0.1$; iii) $a_3 = 0.05$; iv) $b_1 = 0.02$, inspired by discussions in [39].

C. Constructive heuristic

The initial solution is generated with a procedure similar to the constructive phase of GRASP. A number of initial solutions are generated by using the semi-greedy constructive method from [14], which is based on the rule Adaptive Shortest Processing Time (ASPT) [40]. This method evaluates jobs insertion at the end of a machine, see [14] for details. The best solution among those is kept as the initial solution. The number of times the semi-greedy heuristic is used is given by the time limit afforded to the generation of the initial solution. In our experiments, we set this as 1% of the maximum time budget allowed to LA-ALNS, which is the input parameter $t_{\max}$.

D. Insertion and removal heuristics

This section describes the removal and insertion methods performed by ALNS. There are six removal heuristics and six insertion heuristics, which are described next. Before each insertion method, the q removed jobs are shuffled.

1) Random removal: This method removes q jobs randomly from the current solution.

2) Greedy expensive cost removal: This method examines all allocations and stores in a set Y the sum of processing cost and setup time of allocated jobs in current solution. After that, the q most expensive jobs are removed.

3) Semi-greedy expensive cost removal: This method creates a set as defined in the previous method. Then, q jobs are randomly removed from the subset of 20% with highest cost.

4) Random machine removal: This method selects a machine randomly, and then removes the first q jobs from that machine. If the number of jobs on the selected machines is less than q, another machine is chosen randomly until it has q jobs removed.

5) Highest cost machine removal: The machine with the highest cost is selected, and the first q jobs are removed. If the number of jobs on the selected machines is less than q, the next machine with higher cost is selected until the q jobs are removed.

6) Shaw removal: This method is inspired on [41]. Jobs that are considered similar and whose reinsertion might reach to better solutions are removed. The similarity criterion used in this work is the sum of processing and setup costs of a job in its position on the machine. At first, a job j' is randomly selected from a random machine. Then, a relation $R(j', j)$ between job j' and all the other j jobs allocated on all machines is calculated. Then, q similar jobs to job j' are removed.

7) Greedy insertion: This method inserts q jobs in the current solution in a greedy way. On each step, the first job is selected from q removed ones. The cost of its insertion in all positions of all machines is evaluated. Then, the job is inserted in the position on the machine with least cost. These steps are executed until all q jobs are allocated.

8) Semi-greedy insertion: This method is similar to the previous one, but instead of inserting the jobs in the best position (greedy), a random position is selected in a pool of the 20% best locations to insert the current job.

9) Lambda insertion: This method inserts 50% of q jobs randomly using all positions of all machines. The other 50% are inserted by means of greedy insertion.

10) ILS insertion: This method is inspired on the perturbation strategy from AIRP algorithm [14]. For each job j in q set, the following steps are performed: i) a machine i is selected randomly; ii) job j is inserted on the best position of machine i, which means the best position is the one with least cost to that machine.

11) Regretting insertion: This method is based on inserting first those jobs with higher regretting cost. The regretting cost is given by the difference between the cost of the best position to allocate a job in all machines and the cost of the second best position. For all q removed jobs, the regretting cost is performed for all positions on all machines. After that, jobs with higher regretting cost are the first to be inserted.

12) Hungarian insertion: This method uses the Hungarian algorithm [33] to allocate the q removed jobs by means of solving subproblems optimally. The process is operated as follows: i) for a current solution with m machines, the first m removed jobs are selected; ii) the better position to allocate each of those jobs in all m machines is calculated; iii) the best costs of allocating in each machine for each jobs are stored in a square matrix. iv) that matrix is used by Hungarian algorithm to define the optimal allocation. These processes are repeated until it has m jobs without allocation. The remaining jobs are inserted by means of greedy insertion.

E. Local search procedure: RVND

A Random Variable Neighborhood Descent (RVND) method is used as local search procedure [11]. RVND operates local searches randomly, which is different from Variable Neighborhood Descent (VND) [12]. The three neighborhood structures used in RVND are described below.

- **Multiple insertion –** $NS^{MI}(s)$: this move reallocates a job from a machine to another position in the same machine, or reallocates a job in any position in another machine. An example of this move is presented in Figure 5, in which job 4 on machine 2 is transferred to the second position in machine 1.
- **Swap in the same machine –** $NS^{SSM}(s)$: this move changes two jobs in the same machine.
- **Swap between different machines –** $NS^{SDM}(s)$: this move changes two jobs from different machines.

RVND uses three local search methods randomly (FI_{MI}, FI_{SSM}, FI_{SDM}) to exploit the search space. Each local

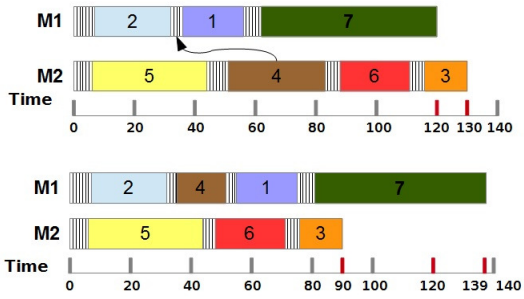


Fig. 5: Example of Multiple insertion

search makes use of the neighborhood structures described previously. Local search operations are presented as follows.

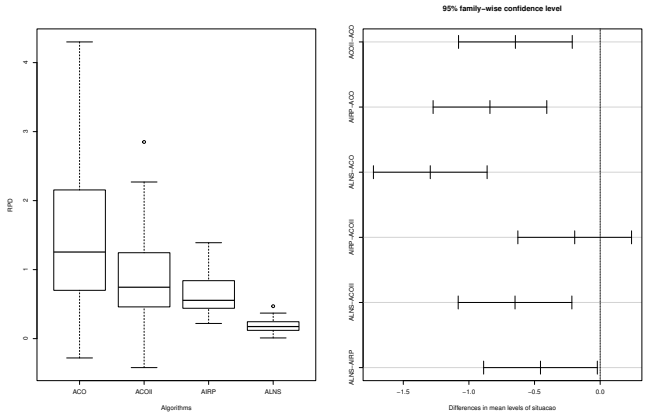
- **Local search** FI_{MI}: this local search is the same one proposed in [14] named FI^1_{MI}. This operation uses $NS^{MI}(s)$ neighborhood structure and first improvement strategy.
- **Local search** FI_{SDM}: this local search is the same one proposed in [29] named Local Search with Swaps Between Different Machines in Section 3.4.2. This operation uses $NS^{SDM}(s)$ neighborhood structure and first improvement strategy.
- **Local search** FI_{SSM}: this local search applies $NS^{SSM}(s)$ neighborhood structure and first improvement strategy. It operates as follows. At first, machines are sorted in descending order by the completion time. Machines are selected from higher to lower completion time. Then, for each machine all changes between its jobs are evaluated. A neighbor $s' \in NS^{SSM}(s)$ is accepted if the completion time of a machine is reduced. The process stops when a solution is accepted; otherwise, it continues.

V. Computational Experiments

Computational experiments are presented in this section. The results of our proposed LA-ALNS are compared to AIRP [14], ACO [15] and ACOII [16]. A subset of 540 large-scale instances was used to compare these algorithms. Theses instances contain problems with 80, 100 and 120 jobs, and 2, 4, 6 and 10 machines. They are divided into three domain-level groups, defined as follows: i) in Balanced group, completion and setup times are balanced; ii) in Process Domain group, the completion times are dominant; iii) in Setup Domain group, the setup times are dominant.

The proposed algorithm has been implemented in Java language with Netbeans 8.0.2 IDE. LA-ALNS and AIRP were executed in a computer with Core i7 processor, 1.9 GHz of clock speed and 6 GB of RAM under Windows 7 operating system. For ACO and ACOII algorithms, it was used the results reported in [16], which were performed in a computer Pentium 4 processor with 2 GB of RAM.

As stopping criterion of LA-ALNS and AIPR, the average execution time of ACOII in [16] was applied. The time was divided by 2.18. This factor was obtained in [42] and it means



(a) Box plot of results (b) Tukey HSD test results.

Fig. 6: Computational results

that our computer is approximately 2.18 times faster than the computer used in [16] for sequential applications. As the clock speed is not available in [16], it was assumed 3.0 GHz to perform comparison because that was compatible with the computer model of the age of the paper.

The comparison of all algorithms was done using the Relative Percentage Deviation (RPD) and is defined by Eq. (3). The result obtained by algorithm *Alg* in instance i is represented by $\bar{f}_i^{Alg}$, and f_i^* is the best solution found in instance i previously. To obtain f_i^*, the LA-ALNS was executed five times for each instance using the same parameters and stop criterion described in section IV. And the best solution out of five runs was stored as f_i^*.

$$RPD_i = \frac{\bar{f}_i^{Alg} - f_i^*}{f_i^*} \times 100 \tag{3}$$

Due to the stochastic feature of algorithms, LA-ALNS and AIRP were executed five times for each instance, and only the mean value is considered. With regard to ACO and ACOII, it was used the results available in [16]. Those algorithms were executed only once for each instance.

Table II presents the mean results of RPD for all algorithms. Each combination of machines and jobs has 15 instances. The values are divided for Balanced, Process Domain and Setup Domain groups, which represents the domain-level of instances.

The best results are highlighted in bold. Negative values indicate that the results of the algorithm were better than the reference value f_i^*. It is clear that the LA-ALNS results were better in most cases, which represents 88% approximately. A box plot of these results is shown in Figure 6a.

Next we present the statistical tests to verify if exists a significant difference between the algorithms. At first, an Analysis of Variance (ANOVA) was performed [43] with 95% of confidence level (threshold $= 0.05$). The value obtained for p-value is equal to 3.32×10^{-11}, therefore it means that exists a significant difference between the algorithms. The assumptions of the test were validated. The Tukey HSD

TABLE II: Results for LA-ALNS, AIRP, ACO and ACOII.

Machines	Jobs	Balanced			
		ACO	ACOII	AIRP	LA-ALNS
2	80	-0.19	**-0.35**	0.44	0.20
	100	-0.04	**-0.31**	0.56	0.12
	120	-0.28	**-0.42**	0.44	0.01
4	80	1.17	0.75	0.84	**0.24**
	100	1.17	0.74	0.91	**0.18**
	120	0.88	0.42	0.81	**0.18**
6	80	2.67	1.68	0.91	**0.30**
	100	2.29	1.48	1.12	**0.20**
	120	1.92	1.02	1.13	**0.26**
10	80	4.30	1.74	1.06	**0.47**
	100	4.23	2.85	1.28	**0.37**
	120	3.44	2.27	1.39	**0.28**

Machines	Jobs	Process Domain			
		ACO	ACOII	AIRP	LA-ALNS
2	80	-0.12	**-0.23**	0.25	0.12
	100	1.83	0.77	0.37	**0.08**
	120	0.78	0.62	0.34	**0.07**
4	80	0.66	0.50	0.49	**0.16**
	100	0.73	0.47	0.55	**0.11**
	120	0.63	0.39	0.34	**0.13**
6	80	1.99	0.51	0.44	**0.37**
	100	1.55	1.22	0.84	**0.19**
	120	1.08	0.76	0.51	**0.16**
10	80	2.82	0.90	0.42	**0.26**
	100	2.34	1.85	0.84	**0.25**
	120	2.02	1.49	0.53	**0.17**

Machines	Jobs	Setup Domain			
		ACO	ACOII	AIRP	LA-ALNS
2	80	-0.06	**-0.16**	0.22	0.12
	100	0.74	0.59	0.34	**0.07**
	120	0.69	0.57	0.32	**0.07**
4	80	0.85	0.64	0.44	**0.12**
	100	0.71	0.45	0.58	**0.13**
	120	0.47	0.35	0.49	**0.09**
6	80	1.81	0.79	0.59	**0.35**
	100	1.75	1.27	0.82	**0.18**
	120	1.34	0.57	0.51	**0.17**
10	80	2.94	0.96	0.65	**0.21**
	100	2.50	1.76	0.79	**0.21**
	120	1.76	1.21	0.58	**0.17**

test [43] was done with 95% of confidence level (threshold $= 0.05$). This test has as objective to identify a significant difference between the results of paired algorithms. Figure 6b shows the result of this test.

The result of Tukey HSD test suggests that LA-ALNS is statistically better than the other algorithms in a pairwise comparison. This result also suggests the efficiency of the proposed algorithm under the defined conditions of the experiment.

Next we present the performance of insertion and removal methods of LA-ALNS to a set of instances. We used the instances with 100 jobs and 10 machines of type Balanced presented in Table II. These instances were executed with time limit of 4.58 minutes. The probabilities of selection were stored every minute. The average probabilities over all instances are shown in Tables III and IV.

It is observed that among the insertion methods the Greedy and Hungarian heuristics had better performance and their probabilities increased over time. Among the removal methods the Highest Cost Machine had the best performance.

TABLE III: Average probabilities of the insertion methods.

	Lambda	Greedy	Semi-Greedy	Hung.	ILS	Regrett.
1min	13.07%	19.68%	18.30%	19.82%	12.86%	16.26%
2min	12.18%	23.24%	18.21%	19.26%	11.99%	15.13%
3min	11.61%	22.67%	16.60%	22.82%	11.88%	14.41%
4min	11.08%	24.78%	15.94%	23.87%	11.13%	13.19%

TABLE IV: Average probabilities of the removal methods.

	Random	Greedy	Semi-Greedy	Highest Cost Machine	Random Machine	Shaw
1min	14.23%	14.79%	14.12%	18.73%	23.69%	14.43%
2min	13.44%	14.17%	14.03%	24.78%	20.28%	13.30%
3min	13.16%	14.55%	14.34%	27.60%	17.78%	12.57%
4min	11.65%	14.69%	13.49%	30.13%	17.73%	12.30%

VI. Conclusions

This work approached the Unrelated Parallel Machine Scheduling Problem with Setup Times, also defined as $R_M|S_{ijk}|C_{\max}$, with the objective of minimizing the makespan. An Adaptive Large Neighborhood Search metaheuristic using Learning Automata to adapt removal and insertion probabilities was proposed to solve the UPMSP-ST efficiently. Six removal methods and six insertion methods were employed to intensify and to exploit the search space, including a new insertion method based on the Hungarian algorithm. Local searches were performed with a Random Variable Neighborhood Descent (RVND) method. The use of Learning Automata on ALNS is an important contribution of this study.

Computational experiments were performed to verify the proposed algorithm, and a set of instances available in [13] was applied. LA-ALNS results were compared to AIRP [14], ACO [15] and ACOII [16]. The experiment suggests that ALNS has found better mean results in 88% of cases. Furthermore, statistical tests were applied to verify if there is a significant difference between the algorithms. The results also suggest that the proposed ALNS has performed statistically better than AIRP, ACO and ACOII under the defined conditions of experiments.

As suggestions for future works, we propose a follow-up study of the application of Learning Automata on the adaptive phase of ALNS. We also intend to incorporate an insertion method using a Mixed Integer Programming (MIP) model to solve subproblems of UPMSP-ST.

VII. Acknowledgements

The authors would like to thank the support given by the Brazilian agencies CAPES, CNPq, and FAPEMIG.

References

[1] X. Zhu and W. E. Wilhelm, "Scheduling and lot sizing with sequence-dependent setup: A literature review." *IIE Transactions*, vol. 38, pp. 987–1007, 2006.

[2] G. Rabadi, R. J. Moraga, and A. Al-Salem, "Heuristics for the unrelated parallel machine scheduling problem with setup times," *Journal of Intelligent Manufacturing*, vol. 17, no. 1, pp. 85–97, 2006.

[3] M. J. Pereira Lopes and J. M. de Carvalho, "A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times," *European Journal of Operational Research*, vol. 176, pp. 1508–1527, 2007.

[4] M. G. Ravetti, G. R. Mateus, P. L. Rocha, and P. M. Pardalos, "A scheduling problem with unrelated parallel machines and sequence dependent setups," *International Journal of Operational Research*, vol. 2, pp. 380–399, 2007.

[5] R. M. Karp, "Reducibility among combinatorial problems," *Complexity of Computer Computations*, vol. 40, no. 4, pp. 85–103, 1972.

[6] M. Garey and D. Johnson, "Computers and intractability: A guide to the theory of NP-completeness," *WH Freeman & Co., San Francisco*, vol. 174, 1979.

[7] R. Graham, E. Lawler, J. Lenstra, and A. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of discrete Mathematics*, vol. 5, no. 2, pp. 287–326, 1979.

[8] O. A. Rosales, F. A. Bello, and A. Alvarez, "Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times," *The International Journal of Advanced Manufacturing Technology*, vol. 76, pp. 1705–1718, 2015.

[9] S. Ropke and D. Pisinger, "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows," *Transportation science*, vol. 40, pp. 455–472, 2006.

[10] T. Feo and M. Resende, "Greedy randomized search procedures," *Journal of Global Optimization*, vol. 6, pp. 109–133, 1995.

[11] M. Souza, I. Coelho, S. Ribas, H. Santos, and L. Merschmann, "A hybrid heuristic algorithm for the open-pit-mining operational planning problem," *European Journal of Operational Research*, vol. 207, no. 2, pp. 1041–1051, 2010.

[12] P. Hansen, N. Mladenovic, and J. A. M. Pérez, "Variable neighborhood search: methods and applications," *4OR: Quartely journal of the Belgian, French and Italian operations research societies*, vol. 6, pp. 319–360, 2008.

[13] SchedulingResearch, "Scheduling research virtual center," 2005, a web site that includes benchmark problem data sets and solutions for scheduling problems. Available at http://www.schedulingresearch.com.

[14] L. P. Cota, M. N. Haddad, M. J. . F. Souza, and V. N. Coelho, "AIRP: A heuristic algorithm for solving the unrelated parallel machine sheduling problem," in *Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC 2014)*, Beijing, 2014, pp. 1855–1862.

[15] J. Arnaout, G. Rabadi, and R. Musa, "A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times," *Journal of Intelligent Manufacturing*, vol. 21, no. 6, pp. 693–701, 2010.

[16] J. P. Arnaout, R. Musa, and G. Rabadi, "A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines - part ii: enhancements and experimentations," *Journal of Intelligent Manufacturing*, vol. 25, pp. 43–53, 2014.

[17] A. C. Glass, C. N. Potts, and P. Shade, "Unrelated parallel machine scheduling using local search," *Mathematical and Computer Modeling*, vol. 20, pp. 41–52, 1994.

[18] B. Srivastava, "An effective heuristic for minimizing makespan on unrelated parallel machines," *Journal of the Operational Research Society*, vol. 49, pp. 886–894, 1997.

[19] M. X. Weng, J. Lu, and H. Ren, "Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective," *International Journal of Production Economics*, vol. 70, pp. 215–226, 2001.

[20] D. W. Kim, K. H. Kim, W. Jang, and F. Frank Chen, "Unrelated parallel machine scheduling with setup times using simulated annealing," *Robotics and Computer-Integrated Manufacturing*, vol. 18, pp. 223–231, 2002.

[21] D. W. Kim, D. G. Na, and F. Frank Chen, "Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective," *Robotics and Computer-Integrated Manufacturing*, vol. 19, pp. 173–181, 2003.

[22] R. Logendran, B. McDonell, and B. Smucker, "Scheduling unrelated parallel machines with sequence-dependent setups," *Computers & Operations research*, vol. 34, no. 11, pp. 3420–3438, 2007.

[23] A. Al-Salem, "Scheduling to minimize makespan on unrelated parallel machines with sequence dependent setup times," *Engineering Journal of the University of Qatar*, vol. 17, no. 1, pp. 177–187, 2004.

[24] K.-C. Ying, Z.-J. Lee, and S.-W. Lin, "Makespan minimisation for scheduling unrelated parallel machines with setup times," *Journal of Intelligent Manufacturing*, vol. 23, no. 5, pp. 1795–1803, 2012.

[25] E. Vallada and R. Ruiz, "A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times," *European Journal of Operational Research*, vol. 211, no. 3, pp. 612–622, 2011.

[26] SOA, 2011, sistemas de Optimizacion Aplicada. A web site that includes benchmark problem data sets and solutions for scheduling problems. Available at http://soa.iti.es/problem-instances.

[27] S. W. Lin and K. C. Ying, "ABC-based manufacturing scheduling for unrelated parallel machines with machine-dependent and job sequence-dependent setup times," *Computers & Operations Research*, vol. 51, pp. 172–181, 2014.

[28] O. A. Rosales, A. Alvarez, and F. A. Bello, "A reformulation for the problem scheduling unrelated parallel machines with sequence and machine dependent setup times," in *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling*, Rome, Italy, 2013, pp. 278–282.

[29] M. N. Haddad, L. P. Cota, M. J. F. Souza, and N. Maculan, "AIV: A heuristic algorithm based on iterated local search and variable neighborhood descent for solving the unrelated parallel machine scheduling problem with setup times," in *Proceedings of the 16th International Conference on Enterprise Information Systems (ICEIS 2014)*, Lisbon, Portugal, 2014, pp. 376–383, DOI: 10.5220/0004884603760383.

[30] L. P. Cota, M. N. Haddad, M. J. F. Souza, and A. X. Martins, "A heuristic algorithm for solving the unrelated parallel machine scheduling problem with sequence dependent setup time (in portugueses)," in *Proceedings of the 35th Brazilian Congress of Applied and Computational Mathematics*, Natal, Brazil, 2014, available at http://proceedings.sbmac.org.br/sbmac/article/view/724/730.

[31] M. N. Haddad, L. P. Cota, M. J. F. Souza, and N. Maculan, "Solving the unrelated parallel machine scheduling problem with setup times by efficient algorithms based on iterated local search," *Lecture Notes in Enterprise Information Systems*, vol. 227, pp. 131–148, 2015.

[32] H. R. Lourenço, O. Martin, and T. Stützle, "Iterated local search," in *Handbook of Metaheuristics*, ser. International Series in Operations Research & Management Science, F. Glover and G. Kochenberger, Eds. Kluwer Academic Publishers, Norwell, MA, 2003, vol. 57, pp. 321–353.

[33] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955.

[34] ——, *The Hungarian Method for the Assignment Problem*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, ch. Chapter 2, pp. 29–47. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-68279-0_2

[35] D. Jungnickel, *Graph Networks and Algorithms (Algorithms and Computation in Mathematics)*, 3rd ed., N. Y. Springer-Verlag, Ed., 2008.

[36] K. S. Narendra and K. S. Thathachar, *Learning Automata: An Introduction*. New York: Prentice-Hall, 1989.

[37] K. S. Narendra and M. A. Thathachar, *Learning automata: an introduction*. Courier Corporation, 2012.

[38] M. M. Alipour, "A learning automata based algorithm for solving capacitated vehicle routing problem," *International Journal of Computer Science Issues*, vol. 9, pp. 138–145, 2012.

[39] R. Vafashoar and M. R. Meybodi, "Multi swarm bare bones particle swarm optimization with distribution adaption," *Applied Soft Computing*, vol. 47, pp. 534 – 552, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1568494616303088

[40] K. R. Baker, *Introduction to Sequencing and Scheduling*. John Wiley & Sons, 1974.

[41] P. Shaw, *Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 417–431. [Online]. Available: http://dx.doi.org/10.1007/3-540-49481-2_30

[42] PassMark, "Cpu benchmarks," 2016. [Online]. Available: https://www.cpubenchmark.net/(accessed1November2016)

[43] D. Montgomery, *Design and Analysis of Experiments*, 5th ed. John Wiley & Sons, New York, NY, 2007.