2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

AIRP: A heuristic algorithm for solving the unrelated parallel machine scheduling problem

Luciano Perdigão Cota, Matheus Nohra Haddad, Marcone Jamilson Freitas Souza and Vitor Nazário Coelho

***Abstract*— This paper deals with the Unrelated Parallel Machine Scheduling Problem with Setup Times (UPMSPST). The objective is to minimize the makespan. In order to solve it, we propose a heuristic algorithm, based on Iterated Local Search (ILS), Variable Neighborhood Descent (VND) and Path Relinking (PR). In this algorithm, named AIRP, an initial solution is constructed using the Adaptive Shortest Processing Time method. This solution is refined by the ILS, having an adaptation of the VND as local search method. The PR method is applied as a strategy of intensification and diversification during the search. The algorithm was tested in instances of the literature envolving up to 150 jobs and 20 machines. The computational experiments show that the proposed algorithm outperforms an algorithm from the literature, both in terms of quality and variability of the final solution. In addition, the algorithm established new best solutions for more than 80,5% of the test problems in average.**

I. Introduction

THIS paper deals with the *Unrelated Parallel Machine Scheduling Problem with Setup Times* (UPMSPST). The objective is to minimize the maximum completion time of the schedule, the so-called *makespan*.

The UPMSPST has great theoretical and practical importance. The practical importance is related to the fact that it appears in many situations, for example, in textile manufacturing [1]. The theoretical importance is due to UPMSPST belongs to the $\mathcal{NP}$-Hard class [2], because it is a generalization of the *Parallel Machine Scheduling Problem with Identical Machines and without Setup Times* [3], [4]. Finding an optimal solution for large problems using exact methods can be computationally infeasible. Therefore, heuristics are generally used to generate solutions that are close, in quality, to the optimal solution.

In UPMSPST there is a set of unrelated machines M and a set of jobs N with the following characteristics: i) Each job must be processed exactly once by one machine; ii) Each job $j \in N$ has a processing time p_{jk}, which depends on the machine $k \in M$ it is allocated; iii) There are setup times, S_{ijk}, between jobs $i, j \in N$, where k is the machine where jobs i and j are processed, in this order; iv) In order to process the first job on each machine, there will be a setup time S_{0jk}, considering the job $j \in N$ allocated on the machine $k \in M$. The objective is to minimize the maximum completion time of the schedule, called makespan or $C_{\max}$. Due to these characteristics, UPMSPST is defined as $R_M|S_{ijk}|C_{\max}$ [5]. In this definition, R_M represents the independent machines, S_{ijk} the setup times and $C_{\max}$ the makespan.

For ease of understanding the problem, in Figure 1 is presented a possible schedule for an instance composed by 7 jobs and 2 machines. The cross-hatched areas of the Figure represent the setup times between jobs and the numbered areas the processing times. It can be observed that in machine 1, called $M1$, the jobs 2, 1 and 7 are allocated in this order. In machine 2, named $M2$, the schedule of the jobs 5, 4, 6 and 3, in this order, is also perceived by this Figure.

TABLE I
PROCESSING TIMES IN MACHINES M1 AND M2.

	M1	M2
1	20	4
2	25	21
3	28	14
4	17	32
5	43	38
6	9	23
7	58	52

TABLE II
SETUP TIMES IN MACHINE M1.

M1	1	2	3	4	5	6	7
1	2	1	8	1	3	9	6
2	4	7	6	3	7	8	4
3	7	3	4	2	3	5	3
4	3	8	3	5	5	2	2
5	8	3	7	9	6	5	7
6	8	8	1	2	2	1	9
7	1	4	5	2	3	5	1

TABLE III
SETUP TIMES IN MACHINE M2.

M2	1	2	3	4	5	6	7
1	3	4	6	5	9	3	2
2	1	2	6	2	7	7	5
3	2	6	4	6	8	1	4
4	5	7	8	3	2	5	6
5	7	9	5	7	6	4	8
6	9	3	5	4	9	8	3
7	3	2	6	1	5	6	7

The processing times of these jobs in both machines are presented in Table I; for example, the cost of processing the job 7 in machine $M1$, p_{71}, is equal to 58.

Luciano Perdigão Cota, Marcone Jamilson Freitas Souza and Vitor Nazário Coelho are with the Department of Computer Science, Federal University of Ouro Preto, Brazil, and Matheus Nohra Haddad is with the Institute of Computation, Fluminense Federal University, Niterói, Brazil (email: {lucianoufop, mathaddad}@gmail.com, marcone@iceb.ufop.br, vncoelho@gmail.com).

This work was supported by the Federal University of Ouro Preto and the Brazilian agencies CAPES, CNPq and FAPEMIG.

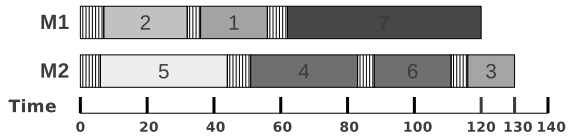

Fig. 1. An example of schedule with 2 machines and 7 jobs.

In Table II and Table III are showed the setup times of these jobs in these machines. For example, the cost of setup time to execute the job 4 after the execution of job 5 in machine $M2$, S_{542}, is the value that is encountered in the fifth line and fourth column, which is 7. Thus, it can be calculated the completion time C_{M1} of machine $M1$ as $C_{M1} = S_{021} + p_{21} + S_{211} + p_{11} + S_{171} + p_{71} = 120$. Equivalently it is also calculated the completion time C_{M2} of machine $M2$ as $C_{M2} = S_{052} + p_{52} + S_{542} + p_{42} + S_{462} + p_{62} + S_{632} + p_{32} = 130$. The makespan will be the completion time of the slowest machine, in this case, the machine $M2$, with a cost of 130.

The objective of this article is to present an efficient algorithm for solving the UPMSPST. The proposed algorithm is based on the heuristic procedures: Iterated Local Search – ILS [6], Variable Neighborhood Descent – VND [7] and Path Relinking [8]. Named AIRP, the algorithm works as follows: i) Build up a solution in a greedy way, using the Adaptive Shortest Processing Time rule – ASPT; ii) Refines this solution by the ILS, using a local search heuristic called RIV, that is inspired in the ILS and Random Variable Neighborhood Descent (RVND) heuristics; iii) Applies the Path Relinking technique to perform a balance between intensification and diversification of the search space. Unlike VND, in RVND there is no fixed sequence of neighborhoods. In [9], the authors demonstrated the efficacy of the RVND over the conventional VND.

The AIRP was tested using instances from literature [10] and the computational results showed that it is able to produce better solutions than another algorithm found in literature, with less variability. In addition, the algorithm established new best solutions for more than 80,5% of the test problems in average.

The rest of this paper is organized as follows: Section II is described the literature review, with related articles. Section III contains the description of the methodology used in this work. Computational results are presented in Section IV to show the AIRP performance. Finally, Section V concludes the paper.

II. Literature Review

Works dealing with similar problems to UPMSPST are found in literature. In [11] seven heuristics are proposed to minimize the weighted mean completion time. In [12], a Simulated Annealing heuristic is used for a problem of minimizing the total completion time. In [13] are proposed four heuristics to minimize the total weighted tardiness. In [14] is addressed a problem where the objective is to minimize the total weighted tardiness considering dynamic releases of jobs and dynamic availability of machines and the authors implemented a Tabu Search for solving it. In [1], the authors proposed a Branch-and-Price algorithm for solving the same problem.

Among the works that address the UPMSPST, more recent references are found. In [15] the authors present a *Three-phase Partitioning Heuristic*, called PH. In [16] it is implemented a *Metaheuristic for Randomized Priority Search* (Meta-RaPS). In [17], the authors implement the *Ant Colony Optimization* (ACO) for a special structure of the problem, wherein the ratio of jobs to machines is large. In [18] it is proposed a method of *Restricted Simulated Annealing* (RSA) which reduces the computational effort by eliminating movements in jobs that will not be effective. The UPMSPST is solved by means of *Genetic Algorithms* in [19]. Using two sets of parameters, the authors implemented two algorithms, GA1 and GA2 and they shown that GA2 outperforms GA1. In this work, test problems for UPMSPST were generated and provided in [10].

III. Methodology

A. Representation of a Solution

The representation of a solution s in UPMSPST is done using a vector of lists. In this representation there is a vector v whose size is the number m of machines. Each position of this vector contains a number that represents a machine and a list of numbers. The schedule of the jobs on each machine is represented by this list of numbers, where each number represents one job.

B. Evaluation of a Solution

The evaluation value of a solution s is the completion time of the machine that will be the last to conclude its jobs, the so-called *makespan*.

C. The AIRP Algorithm

The proposed algorithm, named AIRP, combines the heuristic procedures Iterated Local Search (ILS), Variable Neighborhood Descent (VND) and Path Relinking (PR). The local search used in ILS, named RIV, was developed based on the heuristic procedures ILS and a variation of VND, called Random Variable Neighborhood Descendent (RVND). The pseudo-code of this algorithm is presented in Algorithm 1.

The Algorithm 1 has two input parameters: 1) $timesLevel$, which represents the number of times in each level of perturbation; 2) $executionTime$, the time in milliseconds that limits the execution of the algorithm.

At line 1 occurs the initialization of the variable that controls the time limit, $currentTime$. Next, it creates three empty solutions: the current solution s, the modified solution s' and the solution that will store the best solution found $bestSol$ (line 2). At line 3, the set of elite solutions ($elite$) is created (see subsection III-K).

At line 4, s receives a new solution created by the Adaptive Shortest Processing Time (ASPT) rule (see subsection III-D). Then, this new solution, in s, passes through local searches at line 5, using the RIV heuristic procedure (see subsection

Algorithm 1: AIRP

```
   input : timesLevel, executionTime
   output: bestSol
1  currentTime ← 0;
2  Solution s, s', bestSol;
3  elite ← {};
4  s ← ASPT();
5  s ← RVI(s);
6  bestSol ← s;
7  elite ← elite ∪ {bestSol};
8  level ← 1;
9  Update currentTime;
10 while currentTime ≤ executionTime do
11     s' ← s;
12     times ← 0;
13     maxPerturb ← level + 1;
14     while times < timesLevel do
15         perturb ← 0;
16         s' ← s;
17         while perturb < maxPerturb do
18             perturb ++;
19             s' ← Perturb_ILS(s');
20         end
21         s' ← RVI(s');
22         elite ← update(s');
23         pr ← random(0,1);
24         if pr ≤ 0.05 and |elite| ≥ 5 then
25             el ← random(1,5);
26             if f(elite[el]) ≠ f(s') then
27                 s' ← BkPR(elite [el], s');
28             end
29         end
30         if f(s') < f(s) then
31             s ← s';
32             updateBest(s, bestSol);
33             elite ← update(s);
34             times ← 0;
35         end
36         times ++;
37         Update currentTime;
38     end
39     level ++;
40     se level ≥ 4 então
41         level ← 1;
42     fim
43 end
44 return bestSol ;
```

III-F). Further, line 6, $bestSol$ receives the solution s with possible changes made by RIV. At line 7, $bestSol$ is inserted in the elite set of solutions. After all these steps, the level of perturbations is set to 1 (line 8) and the execution time is recalculated at line 9.

The iterative process of ILS is initiated in lines 10 to 43 and it finishes when the time limit is exceeded. A copy of the current solution to the modified solution is made at line 11.

At lines 12 and 13 the variable that controls the number of times in each level of perturbation ($times$) is initialized, as well as the variable that limits the maximum number of perturbations ($maxPerturb$). The following loop is responsible for controlling the number of times in each level of perturbation (lines 14-38).

The next loop, lines 17 to 20, executes the perturbations (line 19) in the modified solution. The number of times this loop is executed depends on the level of perturbation. With the perturbations accomplished, the new solution obtained is evaluated and the RIV procedure is applied, line 21. Then, at line 22 the elite set is updated taking into account this new solution.

A random real number between 0 and 1 is generated at line 23. If this number is less than 0.05 and the elite set is complete (with 5 solutions), the Path Relinking (see subsection III-K) will be executed using the Backward strategy – BkPR, lines 25 to 28. The base solution used is taken randomly from the elite set and the guide solution is the solution s' returned by the local search RIV. In this way, if the elite set is complete, the BkPR has a 5% chance of being executed.

Between lines 30-35 it is verified if the changes made in the current solution were good enough to continue the search from it. When the time is up, in $bestSol$ will be stored the best solution found by AIRP. When the level, which defines the intensity of the perturbations, is greater than or equal to 4, AIRP restarts its value to 1 (lines 39 to 42).

The following subsections present details of the each module of AIRP.

D. Adaptive Shortest Processing Time

The Adaptive Shortest Processing Time (ASPT) rule is an extension of the Shortest Processing Time rule [20].

In ASPT, firstly, it is created a set $N = \{1, ..., n\}$ containing all jobs and a set $M = \{1, ..., m\}$ that contains all machines.

From the set N, the jobs are classified according to an evaluation function g_k. This function is responsible to obtain the completion time of the machine k. Given a *Candidate List* (CL) of jobs, it is evaluated, based on the g_k function, the insertion of each of these jobs in all positions of all machines. The aim is to obtain in which position of what machine that the candidate job will produce the lowest completion time, that is, the $g_{\min}$.

If the machine with the lowest completion time has not allocated any job yet, its new completion time will be the

sum of the processing time of the job to be inserted with the initial setup time for such job.

If this machine has some job, its new completion time will be the previous completion time plus the processing time of the job to be inserted and the setup times involved, if it has sequenced jobs before or after.

This allocation process ends when all jobs are assigned to some machine, thus producing a feasible solution, s. This solution is returned by the heuristic. The algorithm is said to be adaptive because the choice of a job to be inserted depends on the pre-existing allocation.

E. Neighborhood Structures

In order to explore the solution space, three neighborhood structures are used. These structures are based on swap and insertion movements of the jobs.

1) Multiple Insertion: This movement, which origins the neighborhood $N^{MI}(.)$, consists in reallocating a job from one position of a machine to any possible position in any machines. Considering the example in Figure 1, Figure 2 illustrates the application of this movement, where the job 1 from machine $M1$ is inserted in machine $M2$ before job 3.

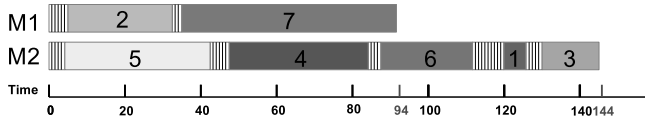

Fig. 2. Example of an insertion movement

2) Swap In The Same Machine: This movement, which origins the neighborhood $N^{SSM}(.)$, consists in swaps two jobs that belong to the same machine. To exemplify, Figure 3 shows the swap of job 2 and 7 from machine $M1$.

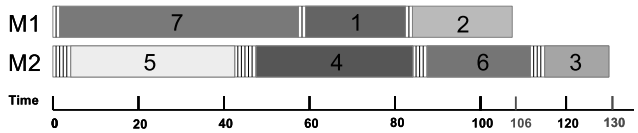

Fig. 3. Example of a swap in the same machine

3) Swap Between Different Machines: This movement, which origins the neighborhood $N^{SDM}(.)$, consists in swaps two jobs that belong to different machines. Figure 4 illustrates the swap between job 7 and 5, from machines $M1$ and $M2$.

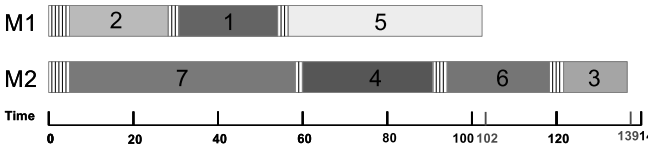

Fig. 4. Example of a swap between different machines

F. RIV Procedure

The RIV procedure is inspired in *Random Variable Neighborhood Descent* – RVND [9] and Iterated Local Search [6]. The pseudo-code of RIV is presented in Algorithm 2.

Algorithm 2: RIV

```
input : s, f(.)
output: s
v ← {1,2,3};
shuffle (v);
k ← 1;
while (k ≤ 3) do
    if k = v[1] then
        s' ← FI^1_MI (s);
    end
    if k = v[2] then
        s' ← BI_SSM (s);
    end
    if k = v[3] then
        s' ← Perturb_SDM (s);
    end
    if f(s') < f(s) then
        s ← s';
        updateBest (s);
        k ← 1;
    end
    else
        k++;
    end
end
Return s;
```

As can be seen, RIV explores the solution space using two local searches and a perturbation method (FI_MI_1, BI_TMM and $Perturb_TMD$) in a random order. The following sections describe these procedures.

G. FI^1_{MI} Local Search

This local search uses the neighborhood $N^{MI}(.)$ through the first improvement strategy. It works as follows: initially, machines are sorted in descending order according to the completion time of each machine. The removals of the jobs are done on machines with higher completion times to the machines with lower completion times. By contrast, the insertions are made from the machines with lower completion times to machines with higher completion times. A neighbor $s' \in N^{MI}(s)$ is accepted if: i) there is a reduction in the cost of the two machines involved (or reduce the cost in the machine involved, if the movement involves a single machine); ii) there is reduction in the cost of one machine and there is an increasing in the cost of the other machine, but overall, it is only accepted if the value reduced is greater than the value increased and the makespan does not increase. This criterion is applied only to movements involving two machines. In case of acceptance, s' becomes the new current

solution s if this procedure is reapplied from the current solution. Otherwise, another neighbor s' is generated. If there is an improvement, reorder up the machines with respect to the cost function and restarts the search. The local search stops when there is no more improvement over the current solution, that is, it is a local optimum with relation to this neighborhood.

H. BI_{SSM} Local Search

This local search uses the neighborhood $N^{SSM}(.)$ through the best improvement strategy. It works as follows: initially, machines are sorted in descending order according to the completion time of each machine. The order of selection of machines is from the machine that has the highest completion time to the machine the has the lowest completion time. Next, it is analyzed for each machine all possible swaps between their jobs. A neighbor $s' \in N^{SSM}(s)$ is accepted if there is a reduction of the completion time of the machine involved. As long as there is improvement, the procedure is repeated from the machine with the highest completion time. Due to the high cost of this local search, this procedure is only applied in 30% of the machines that have the highest completion times.

I. $Perturb\_SDM$ Procedure

This procedure uses the neighborhood $N^{SDM}(.)$. It works as follows: initially, machines are sorted in descending order according to the completion time of each machine. Again, the order of selection of machines is from the machine that has the highest completion time to the machine the has the lowest completion time. Swaps are made from machines that have higher completion times to machines with lower completion times. Next, it is analyzed for each pair of machines all possible swaps between their jobs. A perturbation $s' \in N^{SDM}(s)$ is accepted if occurs reduction in completion time of both machines involved. It is noted that this procedure can generate a solution that has a higher makespan. If the perturbation is accepted, the procedure is stopped; otherwise, the search continues.

J. $Perturb\_ILS$ Procedure

This procedure is characterized by perturbing the local optimum solution by removing a job from a machine and inserting it into another machine. Hence, it is a way of escaping from local optimum and exploring other regions of the search space. Initially it randomly chooses a machine, then it removes a job, taken in a random choice, from this machine. Then, another machine is selected randomly and this job is inserted on its best position in relation to this machine. The best position for the job is the position where the machine will have the lowest completion time with the insertion of this job. In this way, after each perturbation, sub parts of the problem are optimized.

The number of perturbations applied to a solution is controlled by the level of perturbation. A level l of perturbation consists in the application of $l + 1$ insertion movements. In AIRP, the maximum level allowed for the perturbations is set to 3, that is, a maximum of 4 insertion movements will occur.

The perturbation level only is increased after the generation of $timeslevel$ perturbed solutions without improving the current solution. On the other hand, whenever a better solution is found, the level of perturbation is set to its lowest level ($l = 1$).

K. Path Relinking

The Path Relinking (PR) strategy makes a balance between intensification and diversification of the search. Its goal is to explore paths that connect high quality solutions. For this to be done, these high-quality solutions are stored in a set of elite solutions. For a solution s to join it, one of two rules has to be satisfied: i) if s has a smaller makespan than the best solution in the elite set; ii) if s has smaller makespan than the worst solution in the elite set and if s differentiates in 10% of all solutions in the elite set. The adopted criteria of diversity is defined as the percentage of different jobs in the same positions. The purpose of this second rule is to avoid inserting, into the elite set, solutions that are very similar. In AIRP the maximum size of elite set is 5. If the elite set is complete, when a new solution is inserted in it, the worst solution is removed from it.

In possession of the elite set, the paths between the high-quality solutions can be build, from a base solution and toward a guide solution. With this finality, the AIRP algorithm uses the backward strategy (BkPR). The path starts from the base solution, as the best solution, and goes to the guide solution, considered the worst solution. In this work, this strategy is applied on the following solutions: 1) a solution randomly chosen in the elite set; and 2) the local optimum returned by the local search RIV.

To characterize the path is necessary to define an attribute. In this case, it is adopted as attribute the schedule of the jobs in a machine. Initially, the guide solution schedules are inserted into a List of Candidates (LC). At each iteration is analyzed the insertion, in the base solution, of an attribute (schedule) of the solution guide. Next, repetitive jobs in the base solution are eliminated. Furthermore, if the machine of the base solution that receives this schedule has any job different from the schedule of the guide solution, then this job is reallocated to its best position on another machine that had not yet set its attribute of the guide solution. The best position is the one that produces the lowest completion time for the machine. With all the analyzes of the attributes in the guide solution done, is added to the base solution the attribute that has the lowest cost in it. This cost is given by the sum of the costs of each machine in base solution. This modified base solution is then submitted to the local search FI^2_{MI} defined in section III-L.

It is noted that, once inserted an attribute in base solution, this attribute can not be changed. Then, this attribute (schedule) of guide solution is removed from LC. This procedure is repeated until LC is empty.

L. Local Search FI^2_{MI}

This local search uses the neighborhood $N^{MI}(.)$ through the first improvement strategy. It works in a similar way to the local search FI^1_{MI} (see section III-G). The two characteristics that differentiate them are: i) the sole criterion for acceptance is the improvement of the makespan; ii) when a movement is accepted, the procedure is stopped.

M. Efficient Evaluation of the Objective Function

Evaluate an entire solution on every insertion or swap movement requires a lot of computational efforts. In order to make the local search more efficient, it is used a procedure that avoids this situation, by evaluating only the jobs that have been modified. Thus, some additions and subtractions are enough to obtain the completion time of each machine.

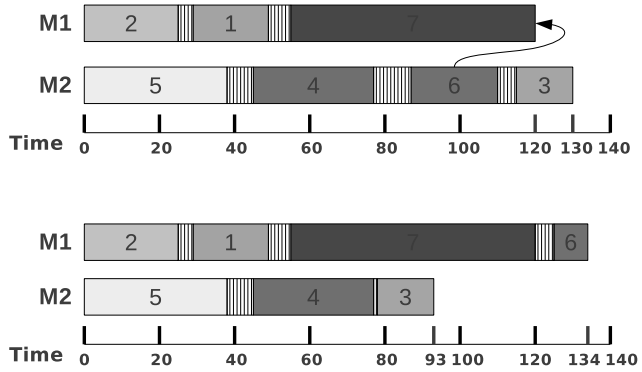

Fig. 5. Evaluating an insertion movement

Figure 5, related to Figure 1, illustrates this procedure for calculating the evaluation function using an insertion movement. The job 6, which was allocated in machine $M2$, is inserted after job 7 on machine $M1$.

The new completion time of machine $M2$ is done subtracting, from its previous value, the processing time of job 6, p_{62}, and also subtracting the setup times involved, S_{462} and S_{632}. It is added, to the completion time of machine $M2$, the setup time S_{432}. The new completion time C_{M2} of machine $M2$ is calculated as $C_{M2} = 130 - 23 - 10 - 5 + 1 = 93$.

In machine M1, are added to its completion time, the processing time of job 6 on this machine, p_{61}, and the setup time S_{761}. Because job 7 is the last to be processed, no setup time is required, so there is no need to subtract anything. The new completion time C_{M1} of machine $M1$ is calculated as $C_{M1} = 120 + 9 + 5 = 134$.

It was given an example of the application of this procedure to evaluate the insertion movement. When dealing with swap movements, the application of this procedure becomes trivial.

IV. Computational Results

Computational tests were performed using a set of 360 test problems from literature, found in [10], involving combinations of 50, 100 and 150 jobs with 10, 15 and 20 machines. For each combination of jobs and machines there are 40 instances. In this site are also provided the best known solutions for each of these test problems.

AIRP was developed in JAVA language and all experiments were executed in a computer with *Intel Core i5 3.0 GHz* processor, 8 GB of RAM memory and in *Ubuntu 12.04* operational system.

The input parameters used in AIRP were: the number of iterations on each level of perturbation, *timeslevel* $= 15$, and the stop criterion was the maximum time of execution $Time_{\max}$, in milliseconds, obtained by Eq. (1). In this equation, m represents the number of machines, n the number of jobs and t is a parameter that was tested with three values for each instance: 10, 30 and 50. It is observed that the stop criterion, with these values of t, was the same adopted in [19].

$$Time_{\max} = n \times (m/2) \times t \;\; ms \tag{1}$$

With the objective to verify the variability of final solutions produced by AIRP it was used the metric given by Eq. (2). This metric is used to compare algorithms. For each algorithm *Alg* applied to a test problem i is calculated the *Relative Percentage Deviation* RPD_i of the solution found $\bar{f}_i^{Alg}$ in relation to the best known solution f_i^*.

$$RPD_i = \frac{\bar{f}_i^{Alg} - f_i^*}{f_i^*} \tag{2}$$

AIRP was compared with the GA2 algorithm from [19] and with the algorithm AIR, which is the AIRP algorithm without the PR method. That is, AIR is the Algorithm 1 without the lines 23-29. The objective of comparing AIRP with AIR is verify the influence of the PR method.

In [19] the GA2 algorithm were executed 5 times for each instance and for each value of t. In this article, the AIRP and AIR algorithms were executed 30 times for each instance and for each value of t, calculating the *Average Relative Percentage Deviation* RPD_i^{avg} of the RPD_i values found.

Table IV shows, for each set of instances, the RPD_i^{avg} obtained for each value of $t = 10, 30, 50$ by the AIRP and AIR algorithms and also by the RPD_i^{avg} values from GA2 algorithm proposed in [19].

For each set of instances three values of RPD_i^{avg} separated by a '/' are found. This separation represents tests results where t values were changed, and the order $t = 10/30/50$ is respected. Negative values indicate that the results reached by AIRP and AIR outperformed the best values found in [19] on their experiments.

In Table IV are highlighted in bold the best values of RPD^{avg}. As noted, AIRP is the one that reached the best results. Not only it wins in all sets of instances, but also it has improved the majority of the best known solutions so far.

It is important to observe that the AIRP algorithm generates final solutions with less variability than the AIR algorithm. This result shows the importance of the Path Relinking method.

TABLE IV
AVERAGE RELATIVE PERCENTAGE DEVIATION OF THE ALGORITHMS AIRP, AIR AND GA2 WITH $t = 10/30/50$.

Set of Instances	AIRP[1]	AIR[1]	GA2[2]
50 × 10	**0.69/-0.51/-0.99**	1.01/-0.3/-0.76	7.79/6.92/6.49
50 × 15	**-2.9/-4.09/-4.63**	-2.69/-3.84/-4.32	12.25/8.92/9.20
50 × 20	**-4.14/-5.26/-5.8**	-3.92/-5.11/-5.56	11.08/8.04/9.57
100 × 10	**1.58/-0.02/-0.59**	1.83/0.26/-0.44	15.72/6.76/5.54
100 × 15	**-1.66/-3.22/-3.94**	-1.48/-3.18/-3.9	22.15/8.36/7.32
100 × 20	**-3.68/-5.51/-6.05**	-3.51/-5.29/-6	22.02/9.79/8.59
150 × 10	**1.29/-0.46/-1.14**	1.3/-0.36/-1.06	18.40/5.75/5.28
150 × 15	**-0.9/-2.61/-3.25**	-0.77/-2.5/-3.19	24.89/8.09/6.80
150 × 20	**-3.95/-5.63/-6.34**	-3.72/-5.52/-6.28	22.63/9.53/7.40
RPD^{avg}	**-1.52/-3.03/-3.64**	-1.33/-2.87/-3.5	17.44/8.02/7.35

[1] Executed on Intel Core i5 3.0 GHz, 8 GB of RAM, 30 runs for each instance
[2] Executed on Intel Core 2 Duo 2.4 GHz, 2 GB of RAM, 5 runs for each instance

The robustness of AIRP can be best seen through the box plot (Fig. 6), which contains all the RPD^{avg} values for each algorithm. It is observed that 100% of the RPD values encountered by AIRP outperforms the ones obtained by GA2 algorithm.

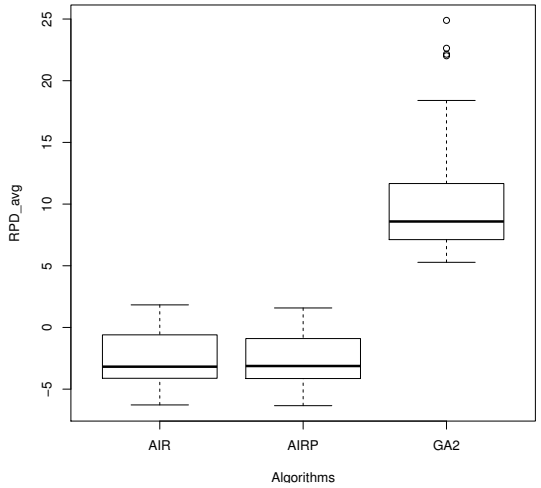

Fig. 6. Box plot showing the RPD^{avg} of the algorithms.

A table with all the results found by AIR and AIRP, and also the previous best know values for the UPMSPST can be found at `http://www.decom.ufop.br/prof/marcone/projects/upmsp/Experiments_UPMSPST_AIR_AIRP.ods`.

According to these complete results, 80,5% of solutions found by AIRP are better than the best known solutions so far.

In order to verify if there is statistical differences between the RPD values, an analysis of variance (ANOVA) [21] was applied. This analysis returned, with 95% of confidence level and $threshold = 0.05$, that $F(2.81) = 115$ and $p = 2 \times 10^{-16}$. As $p < threshold$, there are statistical differences between the RPD values.

To check where are these differences, it was used the Tukey HSD test, with 95% of confidence level and $threshold = 0.05$. Table V contains the differences in the average values of RPD (diff), the lower end point (lwr), the upper end point (upr) and the *p-value* (p) for each pair of algorithms.

It can be seen by the *p-value* that when comparing AIRP and AIR to GA2 there are statistical differences between them because the *p-value* was less than the $threshold$. However, when AIRP is compared to AIR, they are not statistically different from each other, since the *p-value* was greater than the $threshold$.

TABLE V
RESULTS FROM TUKEY HSD TEST.

Algorithms	diff	lwr	upr	p
AIRP-AIR	-0.1633333	-2.592907	2.26624	0.9858998
GA2-AIR	13.5029630	11.010273	15.99565	0.0000000
GA2-AIRP	13.6662963	11.236723	16.09587	0.0000000

By plotting the results from Tukey HSD test (Fig. 7), it can be better seen that AIRP is statistically different from GA2, as the graph do not pass through zero. Also when making a comparison between AIR and GA2, the results show a better performance of AIR.

Comparing algorithms AIRP and AIR it can be perceived that they are not statistically different from each other, because the graph passes through zero. Thus, with a statistical basis it can be concluded, within the considered instances, that both AIRP and AIR are the best algorithms on obtaining solutions for UPMSPST.

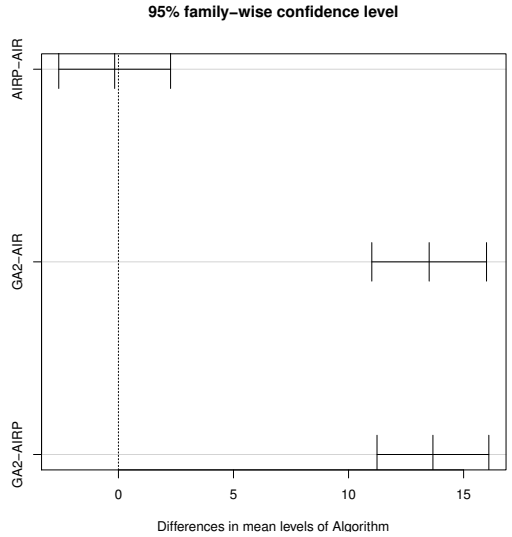

Fig. 7. Graphical results from Tukey HSD test.

V. CONCLUSIONS

This paper addressed the *Unrelated Parallel Machine Scheduling Problem with Setup Times* (UPMSPST). The

desired objective was the minimization of the maximum completion time of the schedule, the so-called *makespan*.

An algorithm based on *Iterated Local Search* (ILS), *Variable Neighborhood Descent* (VND) and Path Relinking (PR) was proposed with the intention to solve it. The algorithm was named AIRP. This algorithm implements the *Adaptive Shortest Processing Time* (ASPT) rule in order to create an initial solution. A procedure inspired on *Random Variable Neighborhood Descent* was used to explore the solution space. This procedure, named RIV, uses two local search methods and a perturbation method in a random order. The first local search method consists of applying multiple insertions of jobs and the second, swap movements of jobs in a same machine. On the other hand, the perturbation method of RIV uses swap movements of jobs in different machines. With the objective of diversifying the search and avoid getting trapped in local optimum, AIRP uses perturbations which consist in reinserting jobs from one machine to another. The PR method is applied periodically as a strategy of intensification and diversification during the search.

By using test problems from literature, AIRP was compared to the genetic algorithm GA2, developed in [19] and with its version without the Path Relinking method, named AIR. Statistical analysis of the computational results showed that both developed algorithms are able to produce better solutions than GA2, setting new best solutions for these test problems. By the experiments, it can be observed that the Path Relinking method contributes to reduce the variability of final solutions of the AIRP algorithm. Thus, it can be concluded that AIRP is a great choice when dealing with the UPMSPST.

As future works, AIRP will be tested on the entire set of test problems available in [10]. An improvement that will be studied is an incorporation of a Mixed Integer Programming (MIP) model to AIRP for solving related sub problems.

REFERENCES

[1] 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.

[2] 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.

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

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

[5] 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.

[6] 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.

[7] 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.

[8] F. Glover, "Tabu search and adaptive memory programming - advances, applications and challenges," in *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, R. S. Barr, R. V. Helgason, and J. L. Kennington, Eds. Kluwer Academic Publishers, 1996, pp. 1–75.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

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