GIVP: A New Algorithm For Solving The Unrelated Parallel Machine Scheduling Problem With Sequence Dependent Setup Times

Abstract

This work presents an algorithm for solving the unrelated parallel machine scheduling problem where setup times are both sequence and machine dependent. The objective is to minimize the maximum completion time of the schedule, the so-called *makespan*. This problem is commonly found in industrial processes, like textile manufacturing, and it also belongs to NP-Hard class. The developed algorithm, named GIVP, is based on *Greedy Randomized Adaptive Search Procedure* (GRASP), *Iterated Local Search* (ILS), *Variable Neighborhood Descent* (VND), *Path Relinking* (PR). To build the initial solution is used the construction phase of GRASP, which is based on the *Adaptive Shortest Processing Time* (ASPT) heuristic. The perturbations of ILS consists in reinserting jobs from one machine to another. The method VND is responsible for conducting the local searches of ILS, exploring neighborhood structures based on swaps and multiple insertions. The PR procedure is applied as an intensification strategy, after the local searches. GIVP was tested using benchmark instances from literature and the computational experiments showed that the results achieved outperformed the results of the literature, both in terms of variability and quality of solutions.

Introduction

The search for efficiency in industrial processes is increasingly intensifying. This is due to the highly competitive fostered by globalization. In order to have efficiency is essential to have a good production planning. Increased productivity and reduced costs are the result of good planning. Computational tools that assist the development of planning are increasingly being used by industries.

Planning the schedule of jobs on multiple machines is a problem constantly found in the industrial sector. The problem consists in defining a sequence of jobs that must be allocated in parallel machines so that the maximum time for completion is minimized. Such machines can be identical or unrelated. They are said to be identical when the processing time to the same job on another machine is identical and unrelated when the processing time to the same job on another machine is different.

Among the various scheduling problems in parallel machines, there is the *Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times*, or, from now on, just UPMSP. This problem is characterized by containing setup times between jobs that are dependent on the machine to which jobs are allocated and dependent on the sequence in which these jobs are allocated. The objective is to minimize the maximum time of completion of the schedule, known as *makespan*.

This problem has both theoretical and practical importance. Theoretical, not only because few studies has been done about this problem, but also because it is a difficult problem that belong to the NP-Hard class (Ravetti et al. 2007). Practical, as there are many situations where it appears, for example, processes in the textile manufacturing (Pereira Lopes and de Carvalho 2007).

Due to the difficulty of finding optimal solutions for UPMSP on acceptable times, the use of heuristics to obtain quality solutions should be considered. In the literature several heuristic approaches that seek to address this problem and similar problems are found. These approaches were the inspiration for this work.

This work proposes a development of a hybrid heuristic algorithm based on *Iterated Local Search* – ILS (Lourenço, Martin, and Stützle 2003) in order to obtain better quality solutions for UPMSP having as optimization criterion the *makespan*. The initial solution is built with the construction phase of *Greedy Randomized Adaptive Search Procedure* – GRASP (Feo and Resende 1995). As an intention to expand the exploration of search space, it is proposed to use the method *Variable Neighborhood Descent* - VND (Mladenovic and Hansen 1997) responsible for conducting local searches in neighborhood structures. It also aims to incorporate into the algorithm the strategy *Path relinking* - PR (Glover 1996) with the objective of intensify and diversify the search in the search space. Not only analysis of the implementation of PR strategy will be made to understand its importance, but also comparisons of the results with those found in the literature.

The rest of this work is structured as follows. Firstly, articles that inspired the development of this work are described. Then, a description of the related problem is done. Next, the methodology used for the deployment of this work is presented. The computational results are shown on se-

quence. Finally this work is concluded and possible proposals to be explored are described.

Literature Review

Some authors address similar problems to UPMSP, only taking into account the setup times dependent on the sequence, not dependent on machines. In Weng, Lu, and Ren seven heuristics are proposed with the objective of minimize the weighted mean completion time. A similar problem is addressed in Kim, Na, and Frank Chen, but with common due dates, and four heuristics are implemented in order to minimize total weighted tardiness. In Logendran, McDonell, and Smucker, the authors used four dispatching rules to generate initial solutions and a *Tabu Search* as the basis for developing six search algorithms, aiming to minimize the total weighted tardiness and also considering dynamic releases of jobs and dynamic availability of machines. A *Branch-and-Price* algorithm is developed in Pereira Lopes and de Carvalho trying to solve the same problem.

Among the works that address the UPMSP more recent references can be found. Al-Salem addresses UPMSP and presents a three-phase partitioning heuristic, called PH. Rabadi, Moraga, and Al-Salem implement a *Metaheuristic for Randomized Priority Search* (Meta-RaPS). This metaheuristic uses a heuristic strategy that combines the construction and refinement, using ideas that are similar to *Greedy Ramdomized Adaptive Search Procedure* (GRASP). In de Paula et al. the method implemented is the *Variable Neighbourhood Search* (VND) to solve unrelated machines problems (UPMSP) and also identical, being imposed due dates for each job and penalties for delays in the due dates.

Arnaout, Rabadi, and Musa implement the method Ant Colony Optimization (ACO) for special structures of the UPMSP, where the ratio of the number of jobs and the number of machines is large. Ying, Lee, and Lin propose a restricted method of *Simulated annealing* (RSA), which reduces the computational effort of the searching by eliminating job movements that wont be effective. Fleszar, Charalambous, and Hindi present a hybrid method of *Variable Neighbourhood Descent* (VND) with mathematical programming.

Vallada and Ruiz try to solve the UPMSP through *Genetic Algorithms*. In the algorithms of these authors, the initial population is generated with one individual created by Multiple Insertion heuristic (MI) (Kurz and Askin 2001) and the rest of the population is randomly generated. Once the population is created, local searches are applied in all individuals. The selection for the crossover is performed by n-tournament. Crossover is made in procedures with limited local searches. Local search based on multiple insertion movements between machines are also applied to all individuals. The selection for survival is made by the stationary strategy, including original individuals in the population if they are better than the worst. Two algorithms, GA1 and GA2, which differ by the parameters were tested. GA2 is the one that had achieved the best results.

Problem Description

In the unrelated parallel machine scheduling problem there is a set $N = \{1, ..., n\}$ of n jobs and a set $M = \{1, ..., m\}$ of m unrelated machines, with the following characteristics:

- Each job must be processed exactly once by only one machine;
- Each job j has a processing time p_{jk} which depends on the machine k where it will be allocated. By this characteristic, the machines are said to be unrelated;
- There are setup times between jobs, s_{ijk}, where k represents the machine whose job i and j will be processed, in that order. These setup times are sequence-dependent and machine-dependent.

The objective is to find a schedule of the n jobs in m machines in order to minimize the maximum completion time of the schedule, called makespan or $C_{\max}$. By the characteristics listed, the UPMSP is defined as $R|S_{ijk}|C_{\max}$ (Pinedo 2008).

To better understand the problem, let a scheduling problem involving seven jobs and two machines. Table 1 contains the processing times of these jobs in both machines, while in Table 2 and Table 3 the setup times of these jobs in these machines are showed. Figure 1 illustrates a possible schedule for this example.

Table 1: Processing times in machines M1 and M2

	1	2	3	4	5	6	7
M1	20	25	28	17	43	9	65
M2	4	21	15	32	38	23	52

Table 2: Setup times in machine M1

M1	1	2	3	4	5	6	7
1	0	1	8	1	3	9	6
2	4	0	7	3	7	8	4
3	7	3	0	2	3	5	3
4	3	8	3	0	5	2	2
5	8	3	7	9	0	5	7
6	8	8	1	2	2	0	9
7	1	4	5	2	3	5	0

Table 3: Setup times in machine M2

M2	1	2	3	4	5	6	7
1	0	4	6	5	10	3	2
2	1	0	6	2	7	7	5
3	2	6	0	6	8	1	4
4	5	7	1	0	12	10	6
5	7	9	5	7	0	4	8
6	9	3	5	4	9	0	3
7	3	2	6	1	5	6	0

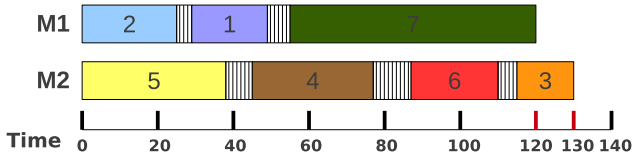

Figure 1: An example of a possible schedule

In Fig. 1 is observed, for example, that job 6 is allocated in third position of machine M2 with job 4 as its predecessor and job 3 as its successor. In Table 1 is showed that job 6 has the processing time in machine M2 (p_{62}) equals to 23. The cross-hatched area of the figure represents the setup times between jobs. Therefore, in this example, the times $s_{462} = 10$ and $s_{632} = 5$ are computed by looking at Table 2. Times marked in red represents the conclusion times of each machine. The conclusion time of machine M1 is 120 and of machine M2 is 130, which results in a makespan of 130.

Proposed Algorithm

In order to resolve the UPMSP, it is proposed an algorithm that combines the heuristic procedures *Greedy Randomized Adaptive Search Procedure* (GRASP), *Iterated Local Search* (ILS), *Variable Neighborhood Descent* (VND) and *Path Relinking* (PR). The construction phase of GRASP is used to build the initial solution, VND is responsible to accomplish the ILS's local searches and PR is used as an intensification and diversification strategy of the searches. The pseudocode of this algorithm, named GIVP, is showed on Algorithm 1.

The Algorithm 1 initializes two solutions on line 1, as well as the set of elite solutions (line 2). Then, on line 3, a solution using GRASP's construction phase is created. This solution passes through local search processes, by means of the VND procedure (line 4). With the result of these local searches, the best known solution, until now, is updated and this solution is inserted in elite set (lines 5-6). The iterative process is situated on lines 8 to 38 and finishes when the stop criterion is satisfied. A copy of the current solution is made on line 9. The following loop is responsible to control the number of times in each level of perturbation (lines 11-33), being this number, $timeslevel$, received as an input of the algorithm. The next loop, lines 15 to 18, execute the perturbations (line 17), so that the number of times the loop is executed depends on the level of perturbation. With the perturbations accomplished, the solution obtained is evaluated on line 19. Then, it is applied the VND procedure in this solution and it is verified if the local optimum reached, in relation to all neighborhoods adopted in VND, can be inserted in elite set (lines 20 and 21). The lines 22 to 26 controls the application of the PR procedure. On lines (27-31) it is verified if the changes made in the solution contributed to a better quality solution. The following subsections presents details of the proposed algorithm.

Algorithm 1: GIVP

```
   Input : timeslevel, stopCriterion
1  Solution s, s';
2  elite ← {};
3  s.createGRASPSolution();
4  s.VND();
5  updateBest(s);
6  elite.insert(s);
7  level ← 1;
8  while stop criterion not satisfied do
9      s' ← s;
10     times ← 0;
11     while times < timeslevel do
12         maxperturb ← level + 1;
13         perturb ← 0;
14         s' ← s;
15         while perturb < maxperturb do
16             perturb ++;
17             s'.perturbation();
18         end
19         s'.evaluate();
20         s'.VND();
21         elite.update(s');
22         pr ← random(0,1);
23         if pr ≤ 0.1 ∧ elite.size ≥ 5 then
24             el ← random(1,5);
25             PR(elite[el], s');
26         end
27         if s'.fo < s.fo then
28             s = s';
29             updateBest(s);
30             elite.update(s);
31         end
32         times ++;
33     end
34     level ++;
35     if level ≥ 4 then
36         level ← 1;
37     end
38 end
```

Representation

A solution s of the UPMSP is represented as a vector of lists. In this representation has a vector v whose size is the number m of machines and each position of this vector contains a number that represents a machine. The schedule of the jobs on each machine is represented by a list of numbers where each number represents the jobs.

For a better understanding of this representation has the example of Fig. 2, where s represents the schedule seen in Fig. 1.

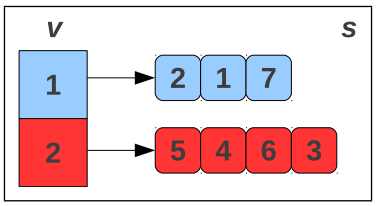

Figure 2: Representation as a vector of lists of the UPMSP

Evaluation of a Solution

A solution s has as evaluation value the processing time of the machine that will be the last to conclude its jobs, this value is called *makespan*.

Initial Solution

To build the initial solution, firstly, an understanding about the *Adaptive Shortest Processing Time* (ASPT) is necessary. In this procedure, first of all, the jobs are sorted in increasing order of processing times, that is, they are ordered with the intention to get the machine on which the job has its shorter processing times. For other jobs, the same sort described above must be performed, however, on machines that have already allocated jobs, should be added, to this time, the setup times. The job is allocated on the machine that has the lowest completion time. This allocation process ends when all jobs are assigned.

The partially greedy construction is done according to the construction phase of GRASP heuristic, using as guide the evaluation function g of heuristic ASPT. On each iteration a Restricted Candidate List (RCL) is built, containing the highest rated jobs of the Candidate List (LC), that is, of the list of job still unallocated. The jobs j of LC are classified according to the guide function g. Jobs j are part of the LRC if $g(j) \leq g_{\min} + \alpha \times (g_{\max} - g_{\min})$, where $g_{\min}$ is the lowest completion time, $g_{\max}$ the highest completion time, and α a GRASP parameter responsible for controlling how greedy the solution will be. Next, a job j is chosen randomly from RCL and allocated to the associated machine. This process is repeated until all jobs are allocated.

Variable Neighborhood Descent

The *Variable Neighborhood Descent* (VND) procedure has as a characteristic the exploration of the space solutions through systematic exchange of neighborhood structures. Such structures are usually associated with predefined order of application. At each iteration of the VND a local search is performed in a neighborhood structure. At the end of this search is obtained the best neighbor of the current solution in relation to this neighborhood structure. If this best neighbor is not better than the current solution, then a new local search in the next neighborhood structure is performed. If this best neighbor is better than the current solution, then the current solution becomes the best neighbor and the local search is restarted at the first neighborhood structure. The procedure ends when no improvement is found in any of the adopted neighborhoods.

Neighborhood Structures

Three neighborhood structures are used in order to explore the solution space. These structures are based on swap and insertion movements of the jobs and they will be described next, in the same order as they are processed by VND. The first neighborhood is analyzed with multiple insertions movements, which are characterized by removing a job from a machine and insert it into a position on another machine, including the machine to which the job was already allocated. The search in the second neighborhood is made by swap movements of the jobs between different machines. The third and final neighborhood is analyzed based on swap movements of the jobs on the same machine. Following, the local searches implemented based on these movements are described.

Local Searches

The first local search uses multiple insertions movements with the strategy *First Improvement*. In this search, each job of each machine is inserted in all positions of all machines. 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. The movement is accepted if the completion times of the machines involved are reduced. If the completion time of a machine is reduced and the completion time of another machine is added, the movement is also accepted. However, in this case, it is only accepted if the value of reduced time is greater than the value of time increased. It is noteworthy that even in the absence of improvement in the value of *makespan*, the movement can be accepted. Upon such acceptance of a movement, the search is restarted and only ends when it is found a local optimum, that is, when there is no movement that can be accepted in the neighborhood of multiple insertion.

The second local search makes swap movements between different machines. For each pair of existing machines are made every possible swap of jobs between them. Exchanges are made from machines that have higher completion times to machines with lower completion times. The acceptance criteria are the same as those applied in the first local search. If there are reductions in completion times on two machines involved, then the movement is accepted. If the reduced value of the completion time of a machine is larger than the completion time plus another machine, the movement is also accepted. Once a movement is accepted, the search stops.

The third local search applies swap movements on the same machine and uses the strategy *First Improvement*. For each machine all possible swaps between their jobs are made. The order of selection of machines is from the machine that has the highest value of completion time to the machine the has the lowest value of completion time. The movement is accepted if the completion time of the machine is reduced. As the acceptance of the movement occurred, the search is restarted and only ends when the local optimum is found in relation to the swap movements on the same machine.

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, is used a procedure that avoids this situation, by evaluating only the jobs environment that have been modified. Thus, some additions and subtractions are enough to obtain the completion time of each machine.

Figure 3 illustrates this procedure for calculating the evaluation function, from an insertion movement. This figure was developed based on Fig. 2 and data tables 1 and 2. It is

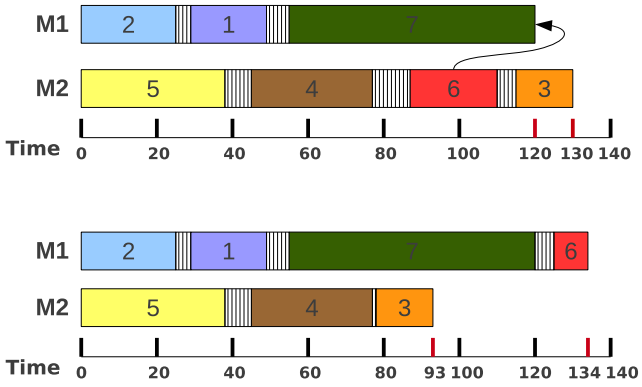

Figure 3: Evaluation on insertion movement

seen the withdrawal of the job 6 from machine M2 and its insertion after job 7 on machine M1. The evaluation procedure calculates the new completion time of machine M2 subtracting from the old value of this 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}. In the machine M1, are added to the completion time, the processing time of job 6 on this machine, p_{61}, and the setup time s_{761}. Because the 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 of machine M1 is calculated by the equation $M1 = 120 + 9 + 5 = 134$ and the new completion time of machine M2 is calculated as $M2 = 130 - 23 - 10 - 5 + 1 = 93$.

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.

Perturbations

The perturbations consist of applying insertion movements on a local optimum. Each perturbation is characterized by removing a job from a machine and inserting it into another machine. By placing the job on another machine, it is searched for the best position for it, that is, the job will be inserted in the position where the machine has the lowest completion time. The choice of machines and the job is done randomly. The amount of modifications in a solution is controlled by a "level" of perturbation, so that a given level n of perturbation consists in the application of $n + 1$ insertion movements. The maximum level allowed for the perturbations is 3. 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 perturbation back to its lowest level.

Path Relinking

The strategy *Path relinking* (PR) 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. This set has two rules for a solution to join. One solution is included in the elite set if it has a smaller value of *makespan* than the value of the best solution already present in it. Also included in the elite set is the solution that is better than the worst solution of the set, but one that satisfy the minimum level of diversity in comparison to other elite solutions. The purpose of this second rule is to avoid inserting, into the elite set, solutions that are very similar. In GIVP algorithm the maximum size of elite set is 5 and the minimum level of diversity is 10%.

The diversity between two solutions is defined as the percentage of different jobs in the same positions. To find the percentage of diversity, a comparison between the jobs for each position of the solution is performed. The sum of the positions that presents different jobs is then divided by the total number of positions of the solution. The result of this division corresponds to the percentage of diversity among the solutions evaluated.

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, it is used the *backward* strategy. Thus, it is considered the best solution as the base solution and the worst solution as the guide solution. In GIVP the base solution is chosen, randomly, as one of the solutions present in the elite set and the guide solution is the solution resulting from application of VND, that is, a local optimum. At each iteration of the PR, a job from the base solution is inserted in the same position that it is encountered in the guide solution. With that done, a local search is performed with multiple insertion movements in this new solution. This procedure is repeated until the base solution is equal to the guide solution. In GIVP, the PR is applied after the execution of the VND, with a probability of 10% and only when the elite set has 5 solutions.

Computational Results

Computational tests were performed using a set of 360 test problems from the literature, found in SOA, involving combinations of 50, 100 and 150 jobs and 10, 15 and 20 machines. For each combination of number of jobs and machines there are 40 instances. In SOA are also provided the best values of *makespan* so far for these test problems.

Two algorithms were implemented, one using GRASP, ILS, VND and PR, named GIVP, and the other without the PR procedure, named GIV. Both were developed in C++ language. All experiments were executed in a computer with *Intel Core 2 Quad 2.4 GHz* processor, 4 GB of RAM memory and in *Ubuntu 10.10* operational system. The parameter used in GIV and GIVP algorithms was: *timeslevel* $= 15$. The stop criterion, of both, was the maximum time of execution $Time_{\max}$, in milliseconds, obtained by Eq. 1, where m represents the number of machines, n the number of jobs and t is the parameter received as an input for the implemented algorithm. The parameter t 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 Vallada and Ruiz.

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

The metric used to compare the algorithms, given by Eq. 2, has the objective to verify the variability of final solutions produced by algorithms. In this metric, it is calculated, for each algorithm *Alg* applied to a test problem i, the Relative Percentage Deviation RPD_i of the found solution $\bar{f}_i^{Alg}$ in relation to the best known solution so far f_i^*. In Vallada and Ruiz the algorithms were executed 5 times for each instance and for each value of t. In this work, the algorithms GIVP and GIV 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.

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

Table 4 shows, for each set of instances, the RPD_i^{avg} for each value of t of algorithms GIV, GIVP, as well as the algorithm GA2 from (Vallada and Ruiz 2011). It is noteworthy that GA2 was executed on computers with similar configurations to those used for the tests, allowing comparisons. 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 outperformed the best values found by Vallada and Ruiz on their experiments.

Table 4: Average Relative Percentage Deviation of the algorithms GIV, GIVP and GA2 with $t = 10/30/50$

Instances	GIV	GIVP	GA2
50 x 10	3.50/2.06/1.50	**-0.84/-1.51/-1.11**	7.79/6.92/6.49
50 x 15	0.42/-1.09/-1.64	**-3.39/-4.76/-4.22**	12.25/8.92/9.20
50 x 20	-0.88/-2.14/-2.89	**-2.71/-4.13/-4.95**	11.08/8.04/9.57
100 x 10	5.43/3.55/2.76	**0.22/-1.77/-1.24**	15.72/6.76/5.54
100 x 15	2.76/0.80/-0.09	**-2.60/-4.21/-4.74**	22.15/8.36/7.32
100 x 20	1.16/-1.43/-2.20	**-5.69/-7.35/-6.65**	22.02/9.79/8.59
150 x 10	5.30/3.43/2.43	**0.47/-1.59/-1.72**	18.40/5.75/5.28
150 x 15	4.36/2.07/1.32	**-2.07/-3.03/-3.85**	24.89/8.09/6.80
150 x 20	1.75/-0.70/-1.62	**-4.16/-6.24/-6.44**	22.63/9.53/7.40
RPD^{avg}	2.65/0.73/-0.05	**-2.31/-3.85/-3.88**	17.44/8.02/7.35

In Table 4 are highlighted in bold the best average values of RPD. As noted, the GIVP algorithm 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 best known solutions so far. The GIV algorithm also reached good results, because when compared to GA2 algorithm, it wins in all sets of instances.

The robustness of GIVP can be best seen through the boxplot (Fig. 4) that contains all the RPD^{avg} for each algorithm. It is observed that almost 100% of the RPD values encountered by GIVP outperforms the best known solutions. Meanwhile, GIV is able to find a little bit more than 25% of solutions that are better than the best ones. Comparing the algorithms it is notable that the worst solutions GIVP can find are better than 50% of solutions produced by GIV. The GIV algorithm, in its turn, has the ability to obtain, near 100%, solutions that are better than solutions obtained by GA2 algorithm. Also, it is important to notice the lower variability of both algorithms developed.

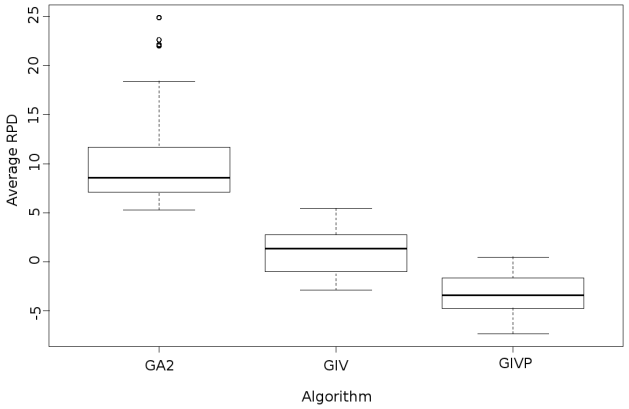

Figure 4: Boxplot showing the RPD^{avg} of the algorithms

In order to verify if exist statistical differences between the RPD values, an analysis of variance (ANOVA) was applied (Montgomery 2007). This analysis returned, with 95% of confidence level and $threshold = 0.05$, that $F(2, 78) = 97.02$ and $p = 2.2 \times 10^{-16}$. As $p < threshold$, exist statistical differences between the RPD values.

To check where are these differences, it was used a Tukey HSD test, with 95% of confidence level and $threshold = 0.05$. Table 5 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-values* that all algorithms are statistically different from each other, because all *p-values* were less than the $threshold$.

Table 5: Results from Tukey HSD test

Algorithms	diff	lwr	upr	p
GIV-GA2	-9.828148	-12.334711	-7.321585	0.0000000
GIVP-GA2	-14.280000	-16.786563	-11.773437	0.0000000
GIVP-GIV	-4.451852	-6.958415	-1.945289	0.0001761

By plotting the results from Tukey HSD test (Fig. 5), it can be seen that all algorithms are statistically different from each other, as their graphs does not pass through zero. The largest difference appears when comparing GIVP with GA2, where GIVP remarkably wins. Also when making a comparison between GIV and GA2 results in a better performance of GIV. Comparing both algorithms developed, GIVP is the one that prevail. Thus, with a statistical basis it can be concluded, within the considered instances, that GIVP is the best algorithm, that is, it is the algorithm that can obtain the best solutions for UPMSP. In addition, it can also be concluded that GIV is better than GA2, generating better

solutions for UPMSP.

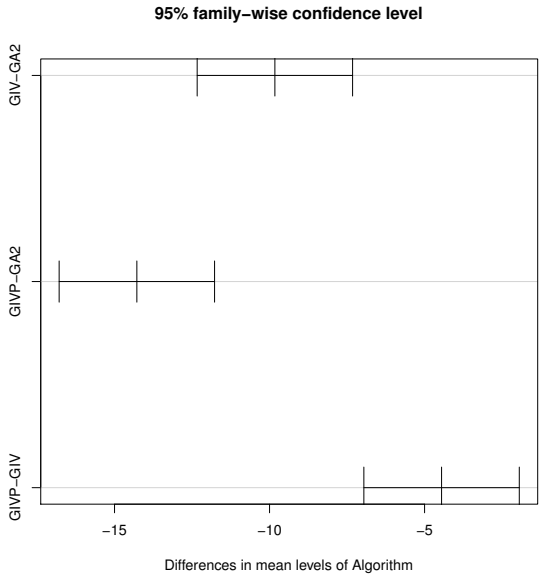

Figure 5: Graphical results from Tukey HSD test

Aiming on a time-based analysis of the developed algorithms, two time-to-target (TTT) plots (Aiex, Resende, and Ribeiro 2002) were created. The use of such plots are important when comparing different algorithms or strategies for solving a problem and have been widely used as a tool for algorithm design and comparison (Feo, Resende, and Smith 1994) (Ribeiro and Resende 2011).

Two experiments were made, in the first experiment the algorithms were applied on a test problem with 100 jobs and 20 machines, and the target was set at 5% of the known optimal value. In the second experiment, the algorithms were applied on a test problem with 150 jobs and 10 machines, and the target was set at 10% of the known optimal value. Both algorithms were executed 100 times and each time the target was reached, they were stopped and the time was stored. These times were then sorted in ascending order, so that, for each execution $i = 1, 2, ..., 100$ there is a time t_i and a probability $p_i = (i - 0.5)/100$ associated. The results of the graph $t_i \times p_i$ are shown in Fig. 6 for the first experiment and in Fig. 7 for the second experiment. For a better comparison between the algorithms, their empirical probability curves were superimposed.

As expected, when analyzing the empirical probability curves (Fig. 6 and Fig. 7) the algorithm with PR procedure (GIVP) prevailed over the one without it (GIV). Since the run time distribution of algorithm GIVP concentrated more on the left part of the graphs, it can be concluded that GIVP finds more quality solutions than GIV in much smaller running times.

In Fig. 6, the algorithm GIVP was the first to reach the desired target, within 25 seconds and with a probability of about 100%. Also, in Fig. 7 the algorithm GIVP was the first to reach the desired target with a probability of about

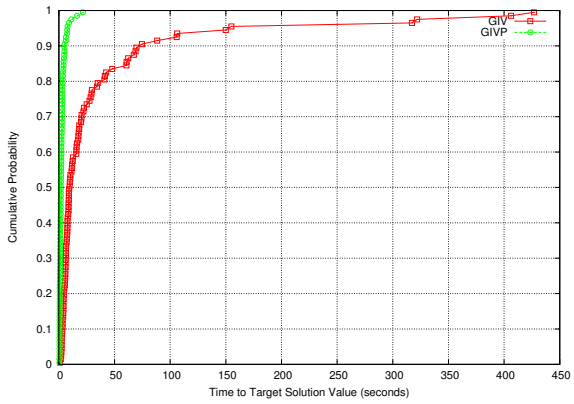

Figure 6: Superimposed empirical distribution - 100 jobs and 20 machines

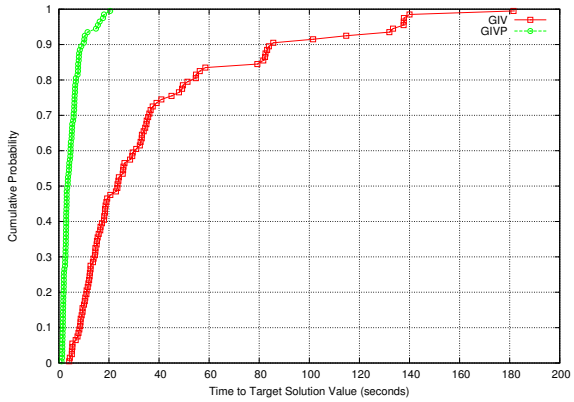

Figure 7: Superimposed empirical distribution - 150 jobs and 10 machines

100% in 20 seconds.

In addition to validate the analysis of the empirical experiments, a probability experiment according to Ribeiro and Rosseti; Ribeiro, Rosseti, and Vallejos was executed. Let A1 and A2 be two stochastic search algorithms applied to the same problem and let X1 and X2, be the continuous random variable that represents the time required for algorithm A1 and A2, respectively, to find a solution as good as the given target. Ribeiro and Rosseti developed a numerical tool to calculate the probability of the runtime of the algorithm A1 be less than or equal to the runtime of the algorithm A2, that is, $Pr(X_1 \leq X_2)$.

This tool was used to calculate the probability mentioned for the algorithms GIVP and GIV, denoted as A1 and A2, respectively. In first test problem with 100 jobs and 20 machines (Fig. 6), the computation shows that $Pr(X_1 \leq X_2) = 90.58\%$, with an error $\epsilon = 0.0357$. In the second test problem with 150 jobs and 10 machines (Fig. 7), the computation shows that $Pr(X_1 \leq X_2) = 94.15\%$, with an error $\epsilon = 0.0005$. In other words, for these two computations, the chance that GIVP reaches a solution as good as the given target is more than 90%.

Conclusions

This paper addressed the unrelated parallel machine scheduling problem where setup times are sequence-dependent and machine-dependent. The desired objective was to minimize the maximum completion time of the schedule, or *makespan*.

An algorithm based on *Iterated Local Search* (ILS) was proposed with the intention to solve it. The perturbations used for this algorithm consist in reinserting jobs from one machine to another. This algorithm implements the construction phase of the *Greedy Randomized Adaptive Search Procedure* (GRASP) in order to create an initial solution based on the *Adaptive Shortest Processing Time* (ASPT) heuristic. The *Variable Neighborhood Descent* (VND) procedure was used to perform the local searches, exploring the solution space with multiple insertions and swap movements. Also were included periodic triggers of the *Path Relinking* (PR) procedure after the local searches, aiming to intensify and diversify the search.

By using test problems from literature, the proposed algorithm, so-called GIVP, was compared to the version without the PR procedure (GIV), as well as to a genetic algorithm from literature. The computational results showed that both versions of the algorithm are able to produce better solutions than the genetic algorithm, with lower variability and setting new lower bounds for these test problems. With a statistical analysis it was showed, within the considered instances, that GIVP is the best algorithm. In addition, the importance of PR, also, could be noticed by the empirical probability test, which showed the improvement in efficiency, because it took less time to reach the target value and with a probability of 90%.

As future work, the algorithms will be tested on the entire set of test problems available in SOA, as well as tested on another sets of test problems available in SchedulingResearch.

References

Aiex, R. M.; Resende, M. G. C.; and Ribeiro, C. C. 2002. Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics* 8:343–373.

Al-Salem, A. 2004. Scheduling to minimize makespan on unrelated parallel machines with sequence dependent setup times. *Engineering Journal of the University of Qatar* 17:177–187.

Arnaout, J.; Rabadi, G.; and Musa, R. 2010. A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing* 21(6):693–701.

de Paula, M. R.; Ravetti, M. G.; Mateus, G. R.; and Pardalos, P. M. 2007. Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighbourhood search. *IMA Journal of Management Mathematics* 18:101–115.

Feo, T., and Resende, M. 1995. Greedy randomized search procedure. *Journal of Global Optimization* 6:109–133.

Feo, T.; Resende, M.; and Smith, S. 1994. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research* 42:860–878.

Fleszar, K.; Charalambous, C.; and Hindi, K. 2011. A variable neighborhood descent heuristic for the problem of makespan minimisation on unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing*. doi:10.1007/s10845-011-0522-8.

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

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

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

Logendran, R.; McDonell, B.; and Smucker, B. 2007. Scheduling unrelated parallel machines with sequence-dependent setups. *Computers & Operations research* 34(11):3420–3438.

Lourenço, H. R.; Martin, O.; and Stützle, T. 2003. Iterated local search. In Glover, F., and Kochenberger, G., eds., *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, Norwell, MA. 321–353.

Mladenovic, N., and Hansen, P. 1997. Variable neighborhood search. *Computers and Operations Research* 24(11):1097–1100.

Montgomery, D. 2007. *Design and Analysis of Experiments*. John Wiley & Sons, New York, NY, fifth edition.

Pereira Lopes, M. J., and de Carvalho, J. M. 2007. A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times. *European journal of operational research* 176:1508–1527.

Pinedo, M. 2008. *Scheduling: theory, algorithms, and systems*. Springer Verlag.

Rabadi, G.; Moraga, R. J.; and Al-Salem, A. 2006. Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing* 17:85–97.

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

Ribeiro, C., and Resende, M. 2011. Path-relinking intensification methods for stochastic local search algorithms. *Journal of Heuristics* 1–22.

Ribeiro, C., and Rosseti, I. 2009. Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. In *Proceedings of the VIII Metaheuristics International Conference*.

Ribeiro, C.; Rosseti, I.; and Vallejos, R. 2011. Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. *Journal of Global Optimization*. Accepted for publication. Available at http://www.ic.uff.br/∼celso/artigos/runtimes.pdf.

SchedulingResearch. 2005. `http://www.schedulingresearch.com`, a web site that includes benchmark problem data sets and solutions for scheduling problems.

SOA. 2011. Test problems for the unrelated parallel machine scheduling problem with sequence dependent setup times. Available at `http://soa.iti.es/problem-instances`.

Vallada, E., and Ruiz, R. 2011. A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *EJOR* 211(3):612–622.

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

Ying, K.-C.; Lee, Z.-J.; and Lin, S.-W. 2010. Makespan minimisation for scheduling unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing*. doi:10.1007/s10845-010-0483-3.