Acta Scientiarum

http://periodicos.uem.br/ojs/acta
ISSN on-line: 1807-8664
Doi: 10.4025/actascitechnol.v40i1.36607

TECHNOLOGICAL INFORMATION

A genetic algorithm and variable neighborhood search for the unrelated parallel machine scheduling problem with sequence dependent setup time

Everton Tozzo, Syntia Lemos Cotrim, Edwin Vladimir Cardoza Galdamez and Gislaine Camila Lapasini Leal*

*Departamento de Engenharia de Produção, Universidade Estadual de Maringá, Avenida Colombo, 5790, 87020-900, Maringá, Paraná, Brazil. *Author for correspondence. E-mail: gclleal@uem.br*

ABSTRACT. This paper presents the evaluation of two metaheuristics to solve the Unrelated Parallel Machine Scheduling Problem with Sequence Machine Dependent Setup Time. Considering such a problem, there is no relation between the time to process each task and the machine; and this is why the machines are referred to as unrelated. Furthermore, the setup time between the executions of two tasks depends on both, the task sequence and its associated machine. A metaheuristic genetic algorithm and a variable neighborhood search were used in order to solve the problem due to the difference among their characteristics. The maximal time for the schedule to be completed, also called makespan, was the performance measure used to evaluate the solutions. The results obtained by both metaheuristics were directly compared according to their performance to try to reduce this makespan. The results showed that the variable neighborhood algorithm search outperformed the genetic algorithm regarding the solutions quality and execution time.

Keywords: scheduling problem; machine scheduling; metaheuristic.

Algoritmo genético e busca em vizinhança variável para o problema de sequenciamento de máquinas paralelas não-relacionadas com setup dependente da sequência

RESUMO. Este artigo avalia a aplicação de duas meta-heurísticas na resolução do problema de escalonamento de máquinas paralelas não relacionadas com o tempo de preparação dependente da sequência e máquina associada. Para esse problema, não existe relação entre a máquina e o tempo de processamento de cada tarefa, por isso as máquinas são denominadas de não relacionadas. Além disso, o tempo de preparação entre a execução de duas tarefas depende de ambos os processos - sequência das tarefas e máquina a elas associadas. As meta-heurísticas algoritmo genético e busca em vizinhança variável foram escolhidas para resolver o problema pela diferença existente entre suas características. A medida de desempenho utilizada para avaliar as soluções é o tempo máximo para conclusão do escalonamento, também denominado *makespan*. Os resultados obtidos por ambas as meta-heurísticas são diretamente comparados de acordo com seu desempenho na tentativa de redução do *makespan*. Os resultados demonstram que a busca em vizinhança variável obteve melhor desempenho comparado ao algoritmo genético no que confere à qualidade das soluções e tempo de processamento.

Palavras-chave: problema de escalonamento; escalonamento de máquinas; metaheurísticas.

Introduction

The task scheduling is an important decision problem at operational level, which appears in different contexts in modern production systems. Its main function is to determine what, when, how and what resources should be used to operationalize the industrial jobs. The sequencing of parallel machines is necessary because several real systems have more than one machine to perform the same set of activities. This issue fits the sub-problems context of greater complexity, such as a flow shop or a job shop, whose set of workstations containing parallel machines works simultaneously. According to Ying, Lee, and Lin (2012), this is a common scenario in current production systems, especially in the textile, chemical, electronics, and implementation services, as well as in the maintenance industry. In practice, this problem is highly complex, not only because of its dimension, but mainly because of the peculiar features arising from the situation to be resolved.

Considering the theoretical aspects, the Parallel Machine Scheduling Problem (PMSP) is considered to be NP-hard, i.e., a polynomial algorithm capable

of generating the optimal scheduling of machines in an acceptable computational time is unlikely to exist.

Sioud, Gravel, and Gagné (2012) consider this challenge as an incentive for the investigation of heuristics that provide good solutions to the problem, which allows the advancement of existing algorithms.

Ravetti, Mateus, Rocha, and Pardalos (2007) point out that the literature on PMSP is extensive. However, the consideration of an environment with unrelated parallel machines is not common, especially when this problem is compared to the amount of publications regarding either identical or uniform machines. Lin, Pfund, and Fowler (2011) emphasize the importance of the algorithms developed for the PMSP with unrelated machines, since it generalizes other problems.

The PMSP can be classified according to the speed to process the tasks in each of the parallel machines. Senthilkumar and Narayanan (2010) classify the problem in three distinct categories: the Identical Parallel Machine Scheduling Problem, the Uniform Parallel Machine Scheduling Problem and the Unrelated Parallel Machine Scheduling Problem.

Several studies on the application of heuristics to evaluate some performance purposes to solve the unrelated PMSP are found in the literature. Glass, Potts, and Shade (1994) conducted a comparison of the metaheuristic genetic algorithms, in addition to simulate an annealing and tabu search to reduce the maximum completion time (makespan). Piersma and Van Dijk (1996) suggested two new local search methods with a neighborhood search referred to as 'efficient' to reduce the makespan. Peyro and Ruiz (2010) also aimed to reduce the makespan by addressing the problem through metaheuristics based on an iterated greedy local search.

Lin et al. (2011) conducted a comparison of the performance of several existing heuristics and proposed a new meta-heuristic for the unrelated PMSP. The authors compared the algorithms from the perspective of three performance purposes analyzed individually and together: 'makespan'; the total weighed completion time and the total weighed tardiness. Lin, Lin, and Hsieh (2013) implemented an ant colony optimization algorithm whose main purpose was to reduce the total weighed tardiness.

As aforementioned, the unrelated PMSP is common in industry, where the manufacturing structure is composed of different machine technologies. However, these differences usually imply not only in distinct processing time for the machines, but also in different preparation time among the tasks according to the associated machine and established sequence for the activities. This is referred to as the unrelated parallel machine-scheduling problem, with a sequence and machine dependent on setup time. According to Vallada and Ruiz (2011), the parallel machine scheduling problem has widely been studied in the last decades when compared to the case involving non-parallel machines. However, the consideration of a sequence dependent on setup time environment has not been applicable until recently.

Kim, Kim, Jang, and Chen (2002) implemented a simulated annealing algorithm for the unrelated PMSP with a sequence dependent on setup time to reduce the total tardiness. The authors consider a variant of the problem where the setup time only depends on the task sequence, but not on the associated machine.

Among the studies that involve the application of metaheuristics for the unrelated PMSP with sequence and machine dependent setup time, Logendran, McDonell, and Smucker (2011) solved the problem with the aim of reducing the total weighed tardiness. The authors considered a dynamic task expedition and a dynamic availability of machines in order to approximate the real situation problem of several industries. Six types of algorithms based on taboo search were implemented. Ravetti et al. (2007) applied a metaheuristics based on GRASP (Greedy Randomized Adaptive Search Procedure) to reduce *makespan* and weighed delays. Paula, Ravetti, Mateus, and Pardalos (2007) used a Variable Neighborhood Search Algorithm (VNS) with the aim of reducing the *makespan* and weighed delays. Chen and Chen (2009) proposed various hybrid metaheuristics by integrating a Variable Neighborhood Descent (VND) and a Taboo Search to reduce the weighed number of tardy jobs. Chen (2009) considered a heuristic to reduce the total tardiness. Vallada and Ruiz (2011) proposed a genetic algorithm to reduce the *makespan* sequencing. Liao, Chang, Kuo, and Liao (2014) reported five hybrid metaheuristics to solve the problem by reducing the *makespan*: three hybrid algorithms by ant colony and two hybrid simulated annealing algorithms.

This paper aims at expanding research on the resolution methods for the unrelated parallel machine scheduling problem, with a sequence and machine dependent on setup time by considering the reduction of the maximum completion time (*makespan*) as a performance measure. Therefore, a direct comparison is carried out between the performance measurements and the results shown by two metaheuristics: a genetic algorithm and a

variable neighborhood search. These metaheuristics were used due to the large difference among their characteristics: the genetic algorithm is classified as a meta-heuristic inspired by nature and based on population, whereas the meta-heuristic VNS is not inspired by nature and performs a punctual search through several neighboring structures. These peculiarities allow a complete diversification of the resolution method for the same problem.

The present study is structured into seven sections, in addition to this introduction. Section 2 characterizes the machine scheduling problem. Section 3 shows the problem representation. Section 4 describes the local search methods, and Section 5 presents the two proposed algorithms to solve the problem. Sections 6 and 7 address experimentation and the results obtained, respectively. Finally, Section 8 shows the final remarks, contributions and considers further research.

Material and methods

A production system with parallel machines is characterized by the availability of a set of m machines, whether identical or not, which executes a set of n tasks in a single production stage. The purpose of the problem is to determine the best sequencing for the tasks, considering not only the set of tasks that should be allocated in each machine, but also the order according to which these tasks must be distributed to improve a particular performance criteria.

The *Unrelated Parallel Machine Scheduling Problem with Sequence and Machine Dependent Setup Time* represents one of the classes derived from the PMSP. In such a case, there is a set $N = \{1, ..., n\}$ of n tasks and a set $M = \{1, ..., m\}$ of m unrelated machines ($m < n$), with the following characteristics (Eroglu, Ozmutlu, & Ozmutlu, 2014):

- Each task must be processed exactly once by only one machine;
- Each task i has a p_{ik} processing time, which depends on the machine k in which it will be allocated. It is due to this characteristic that the machines are referred to as unrelated;
- The setup time depends on both, the sequence of tasks and their associated machine. Let s_{ijk} be the setup time of machine k between processing tasks i and j, in this order. So, the setup time of machine k between tasks i and j is different from the setup time of machine k between tasks j and i, i.e., $s_{ijk} \neq s_{jik}$. In addition, the setup time between tasks i and j in machine k is different from the setup time between tasks i and j in machine h. Therefore $s_{ijk} \neq s_{ijh}$.

Different characteristics may be associated with each of the processing tasks, such as availability date, setup, preemption, precedence constraints, machinery breakdowns, eligibility restrictions, permutations, locks, recirculation, among others.

When n tasks are sequenced in m parallel machines, each machine k has a different time C_k to complete the processing of all the tasks associated with it. The maximum completion time of all machines is known as the *makespan*, represented by $C_{max} = max_{1 \leq k \leq m} \{C_k\}$. The *makespan* is the completion time that will be consumed by the last machine to complete its tasks, also called bottleneck machine (Vallada & Ruiz, 2011). In other words, if C_i is the completion time of job i, so the *makespan* can also be represented by $C_{max} = max_{1 \leq i \leq n} \{C_i\}$.

Given these characteristics, the unrelated PMSP with sequence and machine dependent setup time may be solved by the resolution of the binary decision variable x_{ij}^k, where x_{ij}^k is 1 if job j is immediately processed after job i on machine k and 0 otherwise. In this case, x_{0j}^k represents a job j which is scheduled in the initial position on machine k.

Let C_i be the completion time of job i, C_0 the completion time for a dummy initial job 0, and G a very large positive number. Lin and Ying (2014) formulated this problem as a mixed-integer programming model:

$$Min\ C_{max} = max_{1 \leq j \leq n} \{C_j\} \quad (1)$$

$$s.t$$

$$\sum_{k=1}^{m} \sum_{\substack{i=0 \\ i \neq j}}^{n} x_{ij}^k = 1 \ \forall\ j \in N \quad (2)$$

$$\sum_{j=1}^{n} x_{oj}^k = 1 \ \forall\ k \in M \quad (3)$$

$$\sum_{\substack{i=0 \\ i \neq q}}^{n} x_{iq}^k - \sum_{\substack{j=0 \\ j \neq q}}^{n} x_{qj}^k = 0 \ \forall\ q \in N, \forall\ K \in M \quad (4)$$

$$C_0 = 0 \quad (5)$$

$$C_j - \left[C_i + \sum_{k=1}^{m} x_{ij}^k (s_{ijk} + p_{jk}) + G\left(\sum_{k=1}^{m} x_{ij}^k - 1 \right) \right] \forall\ i = 0, .., n\ \forall j \in N \quad (6)$$

$$x_{ij}^k \in \{0,1\}\ \forall\ i = 0, \dots, n\ \forall\ j \in N\ \forall k \in M \quad (7)$$

Equation 1 represents the objective function of the problem, i.e. the *makespan* minimization. Equation 2 shows that each job is processed once

and by only one machine. Equation 3 ensures that only one job will be assigned to the first position on each machine. Equation 4 guarantees that each job is preceded and succeeded by no more than one job. Equation 5 sets the completion time for the dummy initial job. Equation 6 calculates the completion time of job *j*, ensuring that the completion time of every job is a non-negative value, and that no job should either precede or succeed the same job. Equation 7 states the decision variable x_{ij}^{k} type as a binary.

According to Pinedo (2008), the unrelated parallel machine scheduling problem with sequence and machine dependent setup times, whose purpose is to reduce the maximum completion time of the sequencing, can be denoted by $R \mid S_{ijk} \mid C_{max}$. In this representation, R indicates the unrelated machines, S_{ijk} the setup time (depending on the machine and the task execution order) and C_{max} the *makespan*.

Based on these characteristics, the unrelated parallel machine-scheduling problem with sequence and machine dependent of setup times has the following input data: the number of tasks, the number of parallel machines, the task processing time for each machine and the setup time of each machine while running two consecutive tasks. From the problem input data, a solution is generated by the distribution of all tasks into the existing machines, since all tasks assigned to the same machine have a processing order.

Local search methods

The local search methods have widely been used in applications involving metaheuristics for improving current solution. In order to solve the proposed problem, three local search methods are applied, which are related to a specific neighborhood structure based on either the movement exchange or the relocation of tasks on the same machine or in two different machines. The neighborhood structures are:

(1) *Insertion*: withdrawal of a task to its original position and insertion into a new position in a different machine;

(2) *SwapDM*: position exchange between two tasks *a* and *b* allocated in different machines;

(3) *SwapSM*: position exchange between two tasks *a* and *b* allocated in the same machine.

Figure 1 shows an example of possible movement exchanges to which an individual *i* can be submitted after the application of each of the neighborhood structures in random tasks belonging to the scheduling. The number in each cell represents a scheduled specific task. The lighter cells correspond to the tasks that have changed position due to the movements imposed by each of the neighborhood structures.

Figure 1. Possible movements for the neighborhood structures. Source: This research.

According to each of the neighborhood structures, the following local search methods are applied:

(1) *LocalSearchInsert:* it is based on the insertion neighborhood structure and consists in an analysis of the possibility of removing a task from a machine and allocation it in all positions of the other existing machines;

(2) *LocalSearchSwapDM*: it is based on the *Swap DM* neighborhood structure, i.e., the analysis of the possibility of exchanging the position of two tasks belonging to different machines;

(3) *LocalSearchSwapSM*: based on the *Swap SM* neighborhood structure, it analyses the possibility of exchanging the position of two tasks belonging to the same machine.

All the local search procedures use a strategy referred to as *First Improvement*. It means that they always perform an insertion/exchange movement when the exchange simulation of a task is accepted. For the *LocalSearchSwapSM* procedure, the exchange movements are accepted in case the completion time of the analyzed machine is reduced. For the *LocalSearchInsert* and *LocalSearchSwapDM* procedures, considering each pair of machines, the insertion/exchange movement is accepted in the following cases:

(1) If the new processing time of both machines is reduced;

(2) If the processing time of one machine is reduced and the processing time of the other machine is increased; so the movement is accepted in case the reduced time is higher than the increased time or if the *makespan* value, considering the two machines, suffers no elevation.

Whenever an insertion/exchange movement is accepted (and hence, executed), the search begins the insertion /exchange simulation for the schedule subsequent task. The search only ends when a local optimum is found, i.e., when the insertion/exchange movements are no longer accepted for all neighborhoods.

Proposed algorithm

The use of two metaheuristics was suggested: Genetic Algorithm (GA) and Variable Neighborhood Search (VNS). These metaheuristics were chosen due to the difference between their characteristics, which allows a diversification of the method to solve the problem: GA is a population nature-inspired meta-heuristic, whereas VNS performs a punctual search through various neighborhood structures. Thus, GA deals with multiple solutions at each iteration, favoring diversity, whereas VNS does not.

All parameters adopted for the genetic algorithm and VNS were obtained through empirical experiments. Moreover, with the aim of performing a fair and independent comparison of the differences between the two metaheuristics, strategies were used to implement the algorithms and to analyze the results obtained by them. The algorithms parameters and the mentioned strategies will be described in this section.

Genetic Algorithm (GA)

The main aspects of the Parallel Machines Sequencing Problem (PMSP) is related to the GA specific vocabulary: an individual corresponds to the sequence and distribution of all tasks on existing machines, their genes are each of its tasks, which shall take values belonging to the set $N = \{1, ..., n\}$. Finally, the fitness function consists in the *makespan* of each individual.

Initially, a constructive heuristic was applied to the development of individuals that belong to the initial population. In this heuristic, for each individual, the set of n jobs to be processed in parallel machines are randomly ordered. Then, each task is tested in all positions of all machines (including the positions of the tasks already entered on the same machine) and it is allocated in the position that results the shortest *makespan*. This procedure is performed for each individual until all the initial population is built. The population size defined for the GA proposed was of 50 individuals.

Roulette method was used for selecting the individuals. Let $mak(i)$ be the fitness function that calculates the *makespan* of an individual i and p the size of the GA population. Since the goal of the problem is to reduce the *makespan*, then the function $f(i)$ used to determine the selection probability of an individual i was calculated according to Equation 8.

$$f(i) = \frac{1/_{mak(i)}}{\sum_{j-1}^{p}\left(1/_{mak(j)}\right)} \qquad (8)$$

From the application of Equation 8, all individuals shall have a selection probability that is inversely proportional to its *makespan* value. Ten pairs of individuals are selected for each generation in execution.

Once the ability is evaluated, two individuals are selected (parents) and s the application of the genetic operators begins. The *crossover* is the first operator to be used. This operator is applied with a probability given by a crossing rate of 100% and based on the model proposed by Vallada and Ruiz (2011).

According to Vallada and Ruiz (2011), the objective of the crossing operator is to generate two good individuals (children) from both selected parents. The 'One Point Order Crossover' is one of the most used crossover operators, which is adapted to the parallel machines scheduling problem. For each machine, a p point is randomly selected in *Parent 1*. Every task located from the first position to position p is copied to the respective machine of *Child 1*. Now, all the tasks located from position $p + 1$ to the last position is copied to the respective machine of *Child 2*. After that, all tasks from *Parent 2* that still do not belong to a particular child are sequenced in it. In this process, the task is allocated in the child in the same machine as the parent's. The tasks from *Parent 2* are inserted in the position that results in the lowest processing time for the machine belonging to the child (which may reduce the *makespan* for the individual, thus improving the schedule).

Figure 2 shows an example of the crossover operator considering two parallel machines and eight tasks. In order to facilitate understanding, the setup time between two tasks is not displayed and all tasks are represented with the same size (which does not mean that their processing time is equal).

Both parents are selected and a point p for each machine is randomly assigned to *Parent 1*. In this case $p_1 = 1$ (machine 1) and $p_2 = 3$ (machine 2). Thus, *Child 1* is formed with the tasks belonging to *Parent 1* from position 1 (task 5) of machine 1, and the tasks from the position 1 to 3 (tasks 8, 2 and 3) of machine 2. The *Child 2* is formed with the tasks belonging to the *Parent 1* from position 2 to 3 (tasks 4 and 6) of machine 1, and the task of position 4 and

5 (tasks 1 and 7) from machine 2. Figure 3b shows that the tasks that belong to *Parent 2*, which have not been assigned to any children yet, are inserted into their respective machines. For *Child 1*, tasks 7, 4 and 6 are inserted into the machine 1 and the task 1 on machine 2. For *Child 2*, tasks 2, 3 and 8 are inserted into the machine 1 and task 5 is inserted into the machine 2. Figure 3 represents the final state for both children generated by the crossover operation. It is seen that the tasks from *Parent 2* are inserted in the position that reduces the processing time for the associated machine. For example, for the machine 1 of *Child 2*, the tasks were sequenced in the order (4, 2, 3, 6, 8). The original order in case this insertion method was not applied would be (4, 6, 2, 3, 8).

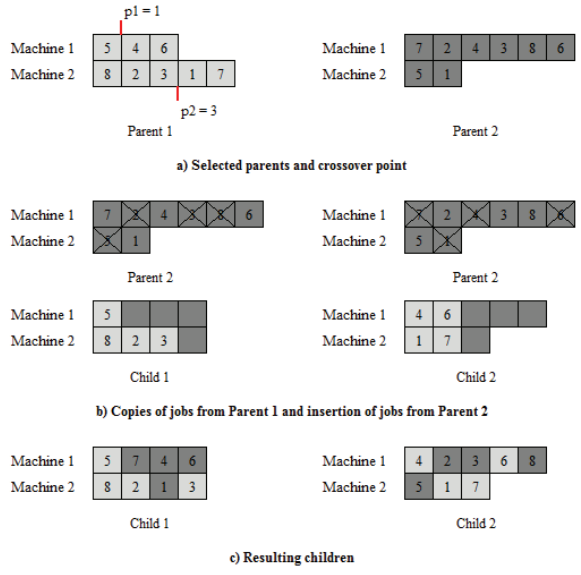

Figure 2. Example of the crossover operator. Source: Vallada and Ruiz (2011).

```
Begin;
1.  s* = s0;                                              // Current solution
2.  While there is improvement on the current solution up to 10 (tem) iterations do:
3.      k = 0;                                            // Type of neighborhood structure
4.      While (k < 3) do:
5.          If (k = 0):
6.              Then s' = InsertMutation (s*)
7.              Else If (k = 1):
8.                  Then s' = SwapMutationDM(s*)
9.                  Else s' = SwapMutationSM(s*)
10.         s'' = VND(s');                                // Local search
11.         If (soluction cost s'' < soluction cost s*):
12.             Then s* = s''; k = 0;
13.             Else k = k + 1;
14. Return s*;
End.
```

Figure 3. Pseudocode for the VNS procedure.

After the crossover operator application, the individuals may suffer some kind of mutation: *InsertMutation, SwapMutationDM* and *SwapMutationSM*. These mutation operators are based on the use of the three neighborhood structures shown above *Insertion*, *SwapDM* and *SwapSM*, respectively. For each mutation operator, its respective neighborhood structure is applied 25 times to the selected individual.

An individual may be subjected to mutation with a probability of 30%. If the mutation occurs, only one of the three operators shall be applied to the individual with a similar choice probability for all of them. A high mutation probability was purposely chosen in order to guarantee more diversity to the population during the evolution of the generations.

After the application of genetic operators, the GA uses three local search methods to improve the current solution: *LocalSearchInsert, LocalSearchSwapDM* and *LocalSearchSwapSM*. A single individual may be sequentially subjected to the three local search procedures. Each procedure is applied with a probability of 50%.

The individuals are reintegrated in the population according to the methodology applied by Vallada and Ruiz (2011): the children generated are accepted only when considered more capable (smaller *makespan*) than the worst individuals in the population. Therefore, generations evolve to better average *makespan* and maintain different solutions, which help to ensure diversity and to avoid premature convergence of the population.

By means of computational tests, it was determined that the best stopping criterion should correspond to the algorithm performance until there is no improvement of the best solution of the population in ten consecutive generations.

Variable Neighborhood Search (VNS)

In order to create the initial solution for the VNS metaheuristics, the best solution generated in the GA initial population was used. This solution already has positive features, once it consists in the best result among a set of p solutions generated from a constructive heuristic, being p the size of the GA population.

This study uses a variant of VNS metaheuristics in which the local search procedure is the Variable Neighborhood Descent (VND) (Hansen, Mladenovic, and Pérez, 2008). The VND uses three neighborhood structures, taken as search intensification mechanisms: *LocalSearchInsert, LocalSearchSwapDM* and *LocalSearchSwapSM*. When a neighbor does not show improvement over the current solution, the next neighborhood structure is used in the local optimum search.

The metaheuristics VNS also uses three neighborhood structures, taken now as search diversification mechanisms: *InsertMutation, SwapMutationDM* and *SwapMutationSM*. These procedures are the same applied to the GA. For each mutation operator, its respective neighborhood structure is applied 25 times to the current solution.

Equivalently to the given GA application, the stopping criterion for the VNS metaheuristics is the performance of ten outer loops of the algorithm with no improvement for the current solution. Let S_0 be the initial solution of the problem, Figure 3 presents a summary of the pseudo-code for the VNS meta-heuristic.

Experimentation

Both proposed algorithms to solve the unrelated parallel machine-scheduling problem, with sequence and machine dependent of setup times were implemented in Pascal and compiled in Lazarus IDE version 1.2.4 environment. The experiments were performed in a machine with four Intel Core i5 processors 2.27 GHz with 4 GB of RAM. There was no parallelism of the experiments code for both algorithms, and Windows 7 was used as the operating system.

As a computational representation of the solution for the algorithms implemented in this study, a $m \times n$ matrix $E = [e_{ij}]$ was used, whose rows represent each of the m parallel machines. The tasks of each machine are inserted sequentially in the columns of the matrix according to the established processing order. When a machine is assigned as t tasks ($t < n$), the subsequent columns ($n - t$) for this machine are filled with numbers 0 (zero), indicating the absence of scheduled tasks.

With the purpose to evaluating the efficiency of the algorithms, both metaheuristics were tested through a standardized set of input data. The processing time for each task (p_{ij}) was defined as integers randomly generated from a uniform distribution U [200, 600], whereas the preparation time (S_{ijk}) considered integers generated from the distribution U [0, 150].

For the problem size, it was generated a set of instances with the combination of the number of machines $m = \{4, 6, 8, 10\}$ and the number of tasks $n = \{50, 75, 100, 150\}$. 100 replications were generated for each combination of machine and task. Each of the instances was submitted to two resolution methods (the GA application and VNS metaheuristics), totaling 3.200 computational tests for comparison of the implemented algorithms.

Results and discussion

With the aim of showing a comparison between the computational results obtained by each of the metaheuristics, the data set averages were analyzed for each combination of machines and and tasks under three related aspects: *makespan* reduction, processing time, and time processing amplitude for each solution.

According to the implementation of both algorithms, the GA and VNS metaheuristics have the same initial solution, which is best individual belonging of the GA initial population. Figure 4 shows the computational results for all instances tested considering the *makespan* reduction. By analyzing Figure 5, it is possible to see that the VNS meta-heuristic achieved a better reduction in *makespan* value from the initial solution, and proved to have a better efficiency in achieving the problem objective.

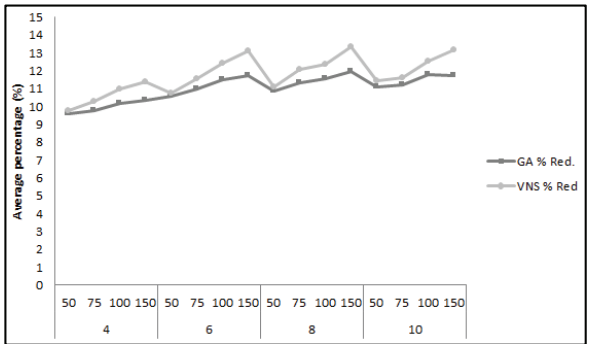

Figure 4. Average percentage of *makespan* reduction.

Figure 5 shows the average time, in seconds, spent by both metaheuristics in order to perform all the application cases. As the initial solution used for the application of VNS metaheuristics corresponds to the best solution found from the previous generation of GA initial population, the time taken to generate the initial population is also considered in the runtime of the VNS algorithm.

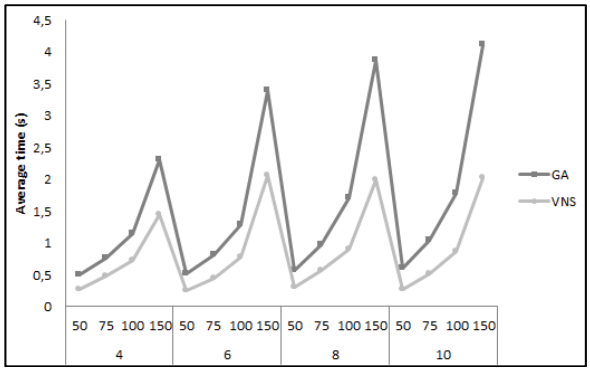

Figure 5. Average processing time.

Based on the analysis of the processing time of both metaheuristics it is seen that the VNS algorithm obtained better results when compared to the GA. Data from Figure 5 show a reducing percentage of the processing time varying from

36.94% to 54.59%. It is reasonable to expect that the processing time of VNS meta-heuristic is lower than the GA due to its characteristics. The GA is based on population, and must process several individuals during each of its generations, which increases processing time.

Considering the complexity of the metaheuristics and the settings of the computer used for experimentation, it is evidences that the time obtained for computational processing enables the practical application of the methods proposed in manufacturing environment.

The graph on Figure 6 shows the average machines processing time amplitude among the machines of the cases tested for both GA and VNS metaheuristics. The solutions range is an important feature to be analyzed, since the scheduling goal is to reduce the *makespan* value, which implies a better distribution of tasks between the machines and, consequentely, closing the processing time of all of them (amplitude tends to zero).

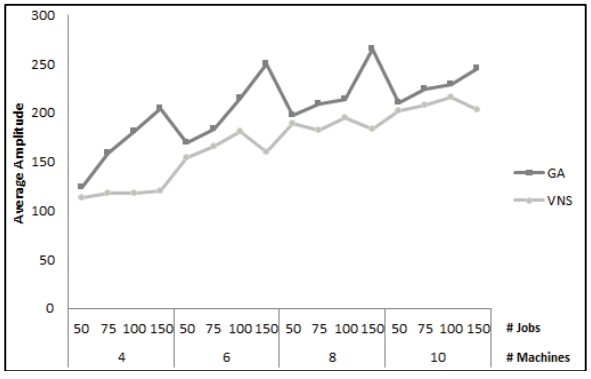

Figure 6. Average machines processing time amplitude for the set of solutions.

Data in Figure 6 reinforce the better efficiency of VNS meta-heuristic when its results are compared with the results obtained by the GA. In all cases, regardless the problem size, the VNS meta-heuristic obtained better amplitudes for the machines processing time. This result is strictly related with the results in Figure 5, i.e., it shows that the lower *makespan* for VNS meta-heuristic is a consequence of a better distribution of tasks in the existing machines, which allows the reduction of the amplitude for the schedule.

The data from Table 1 summarizes the results of the GA and VNS metaheuristics considering a direct and punctual comparison of each of the solutions obtained from the tested instances. Columns four, five and six of the table represent, for the entire set of 100 events, the percentage of cases in which GA obtained lower *makespan*; the percentage of cases in which a VNS metaheuristics obtained lower *makespan;* and the percentage of cases in which the *makespan* found for both methods has the same value, respectively.

Table 1. Ratio for best solution.

#Mach.	#Jobs	Tested Instances	Best Solution		
			%AG	%VNS-VND	%Draw
4	50	100	35	63	2
	75	100	16	84	0
	100	100	3	97	0
	150	100	1	99	0
6	50	100	34	66	0
	75	100	27	71	2
	100	100	7	93	0
	150	100	0	100	0
8	50	100	42	58	0
	75	100	28	71	1
	100	100	17	82	1
	150	100	1	99	0
10	50	100	35	62	3
	75	100	32	67	1
	100	100	19	81	0
	150	100	1	99	0

The data in Table 1 indicate that the VNS meta-heuristic always achieved higher percentage of solutions with lower *makespan* than the GA solutions. Considering the cases in which the problem size has little amount of tasks, the GA achieved a more significant percentage of better solutions. The interpretation of this result is the lack of GA adaptability when different sizes of the problem are considered.

It should be highlighted that the fact that the VNS outperformed GA in large instances for the specific problem evaluated in this study does not mean that VNS is generally better than GA, but rather that the proposed GA algorithm parameters for the chosen problem may not be set appropriately (population size, mutation and crossover probabilities, etc.), what might have caused the GA inefficiency when applied to more complex problems.

Conclusion

The computational results showed different aspects of the solutions obtained by the two algorithms. The VNS meta-heuristic showed a better ability to improve the initial solution and flexibility concerning the problems size. Therefore, these metaheuristics allow a better distribution of existing tasks and reduce the amplitude of processing time of the parallel machines. In addition, the VNS algorithm showed better processing time when compared with GA, showing expressive results and execution time that enable its practical application in an industrial environment.

An evaluation and a test of the two developed metaheuristics are suggested in a further study, which shall consider different configuration parameters, such as the population size, a selection and update method of the GA population; neighborhood structures for the VNS and stopping criteria for both algorithms. Improvement and extensions in the current algorithms should also be considered.

In addition, the application of the proposed methods for solving the unrelated parallel machine scheduling problem is suggested, with sequence and machine dependent setup time, as well as the evaluation of other performance measures, besides *makespan*. Therefore, additional features for scheduling may be coupled to the input data, allowing the investigation of other scheduling aspects, such as the minimization of the maximum delay for the tasks or the quantity of delayed tasks. It means that adjustments in the algorithm may consider multi-objective problems, in which several performance measures are simultaneously taken to analyze the results.

References

Chen, C. L., & Chen, C. L. (2009). Hybrid metaheuristics for unrelated parallel machine scheduling with sequence-dependent setup times. *International Journal of Advanced Manufacturing Technology, 43*, 161-169. doi: 10.1007/s00170-008-1692-1

Chen, J. F. (2009). Scheduling on unrelated parallel machines with Sequence and Machine-dependent setup times and due-date constraints. *International Journal of Advanced Manufacturing Technology, 44*(11-12), 1204-1212. doi: 10.1007/s00170-008-1917-3

Eroglu, D. Y., Ozmutlu, H. C., & Ozmutlu, S. (2014). Genetic algorithm with local search for the unrelated parallel machine scheduling problem with sequence-dependent set-up times. *International Journal of Production Research, 52*(19), 5481-5856. doi: 10.1080/00207543.2014.920966

Glass, C. A., Potts, C. N., & Shade, P. (1994). Unrelated parallel machine scheduling using local search. *Mathematical and Computer Modeling, 20*(2), 41-52. doi: 10.1016/0895-7177(94)90205-4

Hansen, P., Mladenovic, N., & Pérez, J. A. M. (2008). Variable neighbourhood search: methods and applications. *A Quarterly Journal of Operations Research, 6*(4), 319-360. doi: 10.1007/s10479-009-0657-6

Kim, D. W., Kim, K. H., Jang, W., & Chen, F. F. (2002). Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer-Integrated Manufacturing, 18*(3-4), 223-231. doi: 10.1016/S0736-5845(02)00013-3

Liao, T. W., Chang, P. C., Kuo, R. J., & Liao, C. J. (2014). A comparison of five hybrid metaheuristic algorithms for unrelated parallel-machine scheduling and inbound trucks sequencing in multi-door cross docking systems. *Applied Soft Computing Journal, 21*, 180-193. doi: 10.1016/j.asoc.2014.02.026

Lin, C. W., Lin, Y. K., & Hsieh, H. T. (2013). Ant colony optimization for unrelated parallel machine scheduling. *The International Journal of Advanced Manufacturing Technology, 67*(1-4), 35-45. doi: 10.1007/s00170-013-4766-7

Lin, S., & Ying, K. (2014). ABC-based manufacturing scheduling for unrelated parallel machines with machine-dependent and job sequence-dependent setup times. *Computers & Operations Research, 51*, 172-181. doi: 10.1016/j.cor.2014.05.013

Lin, Y. K., Pfund, M. E., & Fowler, J. W. (2011). Heuristics for minimizing regular performance measures in unrelated parallel machine scheduling problems. *Computers and Operations Research, 38*(6), 901-916. doi: 10.1016/j.cor.2010.08.018

Logendran, R., McDonell, B., & Smucker, B. (2011). Scheduling unrelated parallel machines with sequence-dependent setup times. *Computers and Operations Research*, 81(9-12), 1487-1496. doi: 10.1016/j.cor.2006.02.006

Paula, M. R., Ravetti, M. G., Mateus, G. R., & Pardalos, P. M. (2007). Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighbourhood search. *IMA Journal of Management Mathematics, 18*(2), 101-115. doi: 10.1093/imaman/dpm016

Peyro, L. F., & Ruiz, R. (2010). Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research, 207*(1), 55-69. doi: 10.1016/j.ejor.2010.03.030

Piersma, N., & Van Dijk, W. (1996). A local search heuristic for unrelated parallel machine scheduling with efficient neighborhood search. *Mathematical and Computer Modelling, 24*(9), 11-19. doi: 10.1016/0895-7177(96)00150-1

Pinedo, M. L. (2008). *Scheduling: theory, algorithms, and systems* (3rd ed.). Springer, OK: Springer.

Ravetti, M. G., Mateus, G. R., Rocha, P. L., & Pardalos, P. M. (2007). A scheduling problem with unrelated parallel machines and sequence dependent setups. *International Journal of Operational Research, 2*(4), 380-399. doi: 10.1504/IJOR.2007.014169

Senthilkumar, P., & Narayanan, S. (2010). Literature review of single machine scheduling problem with uniform parallel machines. *Intelligent Information Management, 2*(8), 457-474. doi: 10.4236/iim.2010.28056

Sioud, A., Gravel, M., & Gagné, C. (2012). A hybrid genetic algorithm for the single machine scheduling problem with sequence-dependent setup times.

Computers & Operations Research, 39(10), 2415-2424. doi: 10.1016/j.cor.2011.12.017

Vallada, E., & Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research, 211*(3), 612-622. doi: 10.1016/j.ejor.2011.01.011

Ying, K. C., Lee, Z. J., & Lin, S. W. (2012). Makespan minimization for scheduling unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing, 23*(5), 1795-1803. doi: 10.1007/s10845-010-0483-3

Received on April 7, 2017.
Accepted on June 13, 2017.