International Journal of Industrial Engineering Computations 6 (2015) 145–156

Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

An ordered heuristic for the allocation of resources in unrelated parallel-machines

André Serra e Santos[a*], Ana Maria Madureira[b] and Maria Leonilde R. Varela[a]

[a]*Department of Production and Systems - School of Engineering - University of Minho (EEUM/UM), Portugal*
[b]*GECAD - Knowledge Engineering and Decision Support Research Center - School of Engineering – Polytechnic of Porto (ISEP/IPP), Portugal*

C H R O N I C L E

Article history:
Received November 12 2014
Received in Revised Format November 15 2014
Accepted January 7 2015
Available online January 9 2015

Keywords:
Scheduling
Makespan
Unrelated Parallel Machines
MCT
MOMCT

A B S T R A C T

Global competition pressures have forced manufactures to adapt their productive capabilities. In order to satisfy the ever-changing market demands many organizations adopted flexible resources capable of executing several products with different performance criteria. The unrelated parallel-machines makespan minimization problem (Rm||Cmax) is known to be NP-hard or too complex to be solved exactly. In the heuristics used for this problem, the MCT (Minimum Completion Time), which is the base for several others, allocates tasks in a random like order to the minimum completion time machine. This paper proposes an ordered approach to the MCT heuristic. MOMCT (Modified Ordered Minimum Completion Time) will order tasks in accordance to the MS index, which represents the mean difference of the completion time on each machine and the one on the minimum completion time machine. The computational study demonstrates the improved performance of MOMCT over the MCT heuristic.

1. Introduction

Operations scheduling is a decision process that defines strategies from orders release down to detailed programming (Baker & Trietsch, 2009; Blazewicz et al., 2001; Pinedo, 2012). Therefore, for determining how an order will be executed it is necessary to decide when an order should be processed and which resource should be used for each task. Moreover, the establishment of a production sequence indicating the tasks' execution order has also to be considered in order to optimize one or more performance measures. Consequently, scheduling tries to answer two main questions (Baker & Trietsch, 2009; Pinedo 2012; Xhafa, & Abraham, 2008):

(1). How should tasks be distributed between the available resources in order to optimize the performance measure(s)?

(2). In which sequence should the tasks be processed to optimize the performance measure(s)?

According to these two main questions, Operations Scheduling can be divided into two main phases, the allocation phase that answers the first question and the sequencing phase that answers the second one. There is a strong relation between both phases and it is often hard to clearly distinguish between them.

* Corresponding author. Tel:+351-919231006
E-mail: 1070953@isep.ipp.pt (A. Serra e Santos)

doi: 10.5267/j.ijiec.2015.1.001

In problems where the resources are in parallel, the allocation phase is usually addressed before determining the sequence in which the tasks will be processed. In problems with more complex implementations, it is harder to separate the allocation phase from the sequencing one, since there is a correlation between them.

In complex production implementations, the problem constraints and the number of available tasks can make scheduling operations extremely hard through exact methods (Baker & Trietsch, 2009; Blazewicz et al., 2001; Pinedo, 2012). Therefore, many different kinds of contributions exist for dealing with these more complex scenarios, namely web-based systems and platforms, which would integrate a variety of scheduling approaches and methods, mainly based on heuristic, for solving different types of scheduling problems. Some examples are described by Varela et al. (2003, 2008, 2012, 2014), which include several distinct kinds of exact and heuristic scheduling methods. This paper proposes an ordered approach to the *MCT* heuristic for unrelated parallel-machines *makespan* minimization problem (*Rm*||*Cmax*) (Pinedo, 2012; Varela et al., 2008), based on the performance limitations of the random like order in which *MCT* allocates tasks. The proposed *MOMCT* is a constructive heuristic, which would construct a schedule from an empty solution by adding parts to the solution (Briceño, 2008).

In this paper, resources are identified as machines, which execute jobs or tasks, and where the elementary part of the job or task only requires a single resource is referred as an operation. The scheduling problem will be represented by the Graham's classification as described by Pinedo (2012) and Varela et al. (2008). This classification is represented by three fields, **α|β|γ** with the first field representing the machine environment, the second field the process characteristics and constraints and the third field the problem optimization criterion. This paper is divided into five sections. Section 2 defines the *makespan* minimization problem in parallel-machines (*Pm*||*Cmax*), in parallel-machines with different speeds (*Qm*||*Cmax*) and in unrelated parallel-machines (*Rm*||*Cmax*). Section 3 presents the literature review and four heuristics for the *Rm*||*Cmax* problem. In section 4, the proposed *Modified Ordered Minimum Completion Time* heuristic is described. The computational study and a discussion of the achieved results is presented in section 5. Lastly, the paper presents some conclusions.

2. Problem Definition

The *makespan* minimization problem in parallel-machines (*Pm*||*Cmax*) is known to be NP-hard. It can describe a machine environment where there are *m* similar machines in parallel. It is equivalent to the PARTITION problem (Pinedo 2012). *Pm*||*Cmax* is solved by the priority rule *LTP* (Longest *Processing Time)*. The problem (*Pm*||*Cmax*) is formulated as seen bellow. In this problem the optimization criterion is to minimize the *makespan* (*Cmax*), which is subjected to three constraints, where *n* is the number of tasks and *m* the number of machines. The first (1) forces each task to be allocated to only one machine; the second (2) defines the *makespan* as the completion time of the last task to leave the system. The third (3) forces the decision variables to be non-negative. *xij* are the decision variables that define which tasks (j) are allocated to each machine (i), p_{ij} is the processing time of task *j* in machine *i*.

min C_{max}

subject to

$$\sum_{i=1}^{m} x_{ij} = 1, \qquad j = 1, \cdots, n \tag{1}$$

$$\sum_{j=1}^{n} p_j x_{ij} \le C_{\max}, \qquad i = 1, \cdots, m \tag{2}$$

$$x_{ij} \ge 0 \tag{3}$$

The *makespan* minimization problem in parallel-machines with preemptions (*Pm|prmp|Cmax*) can be solved exactly since it is possible to divide the execution of tasks among the various machines. *Pm|prmp|Cmax can* also be solved by the priority rule *LRPT* (Longest *Remaining Processing Time)*. The problem (*Pm|prmp|Cmax*) is formulated as seen bellow. In the problem the optimization criterion is to minimize the *makespan* (*Cmax*), which is subjected to four constraints. Eq. (4) forces each task to be executed, Eq. (5) makes sure that each task execution is less than or equal to the *makespan.* Eq. (6) ensures that the processing time in each machine is less than or equal to the *makespan.* Eq. (7) forces the decision variables to be non-negative.

min C_{max}

subject to

$$\sum_{i=1}^{m} x_{ij} = p_j, \qquad j = 1,\cdots,n \tag{4}$$

$$\sum_{i=1}^{m} x_{ij} \le C_{\max}, \qquad j = 1,\cdots,n \tag{5}$$

$$\sum_{j=1}^{n} x_{ij} \le C_{\max}, \qquad i = 1,\cdots,m \tag{6}$$

$$x_{ij} \ge 0 \tag{7}$$

The *makespan* minimization problem in unrelated parallel- machines (*Rm||Cmax*) is a particular case of *makespan* minimization problem in parallel-machines (*Pm||Cmax*) (Pinedo, 2012). In this problem, the machine has different characteristics, which means, each machine can have a different processing time for each task. Unrelated parallel-machines can describe problems in workshops where there are different machines that can be used to execute the same tasks, which is usual when considering an upgrade to the installed capacity, or grid-computing problems where computers with different configurations are shared between various users. The problem (*Rm||Cmax*) formulation can be seen bellow.

min C_{max}

subject to

$$\sum_{i=1}^{m} x_{ij} = 1, \qquad j = 1,\cdots,n \tag{8}$$

$$\sum_{i=1}^{n} p_{ij} x_{ij} \le C_{\max}, \qquad i = 1,\cdots,m \tag{9}$$

$$x_{ij} \ge 0 \tag{10}$$

In this problem the optimization criterion is to minimize the *makespan* (*Cmax*), which is subjected to three constraints, where *n* is the number of tasks and *m* the number of machines. Eq. (8) forces each task to be allocated to only one machine, Eq. (9) defines the *makespan* as the completion time of the last task to leave the system. Eq. (10) forces the decision variables to be non-negative. Finally, x_{ij} are the decision variables that define which tasks (j) are allocated to each machine (i), p_{ij} is the processing time of task *j* in machine *i*.

3. Literature Review

Due to the complexity of the $R_m||C_{max}$ problem, many heuristics have been developed. Some of the most used algorithms, presented in (Ibbara et al. 1997), such as the, *MCT Min-Min* and *Min-Max* heuristics are efficient methods to approach the problem. Braun et al. (2001) presented and compared *OLB* (*Opportunistic Load Balance*), *MET* (*Minimum Completion Time*), *MCT*, *Min-Min*, *Min-Max*, *Duplex*, *GA* (*Genetic Algorithms*), *SA* (*Simulated Annealing*), *GSA* (*Genetic Simulated Annealing*), *Tabu Search* and *A**. In the computational study, *GA* had the best performance and *Min-Min* the best results between the constructive heuristics. *OLB* and *MET* constructive heuristics had the worse results. Other approaches (Briceño et al., 2012; Chaturvedi & Sahu 2011; Pfund et al., 2004) to the problem are the Suffrage heuristics that allocates the tasks that would suffer more when not allocated to their favorite machine first, *KPB* (*K-Percent Best*) and *SWA* (*Switching Algorithm*) that combine the *MET* and *MCT* heuristics, *WQ* (*Work Queue*) that allocates task to the machine with the minimum work load. *DFPLTF* (*Dynamic Fastest Processor to Largest Task First*) and *RR* can be used to approach the dynamic $R_m||C_{max}$ problem. Sivasankaran et al. (2010) proposed a local search heuristic. The proposed heuristic consists of two phases: in the first tasks are allocated to the machines and in the second they are shifted from one machine to another. Sugavanam et al., (2007) compared *SA* to *GRASP* (*Greedy Randomized Adaptive Search Procedure*). The computational study found that the *SA* performed better than the *GRASP*. Some research is focused on mixed approaches, which in the first phase use integer programing and in the second phase use heuristics methods. Hariri et al. (1991) proposed the *DA* (*Descent Algorithm*) and Glass et al. (1994) proposed an altered *GA,* which applies the *DA* to each population solution, the *GDA* (*Genetic Descent Algorithm*) is then compared with *DA*, *SA*, *TS* and *GA*. In the computational study the *GDA* outperformed the *GA*, with results similar to those of *SA* and *TS*. The *LP/Roundup* heuristic is proposed by Lin et al. (2011), the heuristic solves the LP and then allocates the task to the machines where they have the highest fraction. In the computational study, the *LP/Roundup* was compared with *MCT*, *LP/MCT*, *LP/M* and *GA*. *LP/Roundup* performed better than the other constructive heuristics, *GA* achieved the best results but at the expense of more computational time.

Lenstra et al. (1990) proposed an approximation algorithm based on LP-relaxation and Martello et al. (1997) proposed lower bounds based on Lagrangian relaxations to obtain new exact and approximation algorithms. Ebenlendr et al. (2014) and Verschae et al. (2014) proposed a special case of the $R_m||C_{max}$ problem. In this case each task needs to be allocated to at most two machines and with the similar execution times (*Restrictive Assignment*), with a two phase *LP/Rounding* approach. Vakhania et al. (2014) proposed another special case of the $R_m||C_{max}$ problem, where each task can have two possible execution (p and q) times. The authors presented a polynomial-time algorithm for $q=2p$ and then modified to deliver approximated solutions for the $p\neq 2p$ problem.

In this section, three heuristics for $R_m||C_{max}$ will be presented, along with an illustrative example with three machines (m=3) and four tasks (n=4), seen in Table 1.

Table 1
Illustrative Example

	Machine 1	Machine 2	Machine 3
Task 1	100	120	260
Task 2	100	170	270
Task 3	200	300	400
Task 4	160	80	300

3.1.Minimum Execution Time

The *MET* (*Minimum Execution Time*) heuristic uses the execution time to allocate tasks. This does not take into consideration tasks already allocated, what can result in an inefficient solution when several tasks are allocated to one machine (Braun et al., 2001; Briceño et al., 2012; Ibbara et al., 1997). The MET algorithm is:

1. Unmapped task are placed into a list in a random sequence;
2. The first task is allocated to the minimum execution time machine;
3. Unmapped task are placed into a list in a random sequence;
4. Steps II, III and IV are repeated until all tasks have been mapped.

In the illustrative example, *MET* found a solution (Fig.1) with *makespan* of 400 *t.u*, with the allocation of tasks 1, 2 and 3 into machine 1; the remaining task (task 4) was allocated into machine 2. The solution is far from the optimal solution that would have a *makespan* of 260 *t.u*.

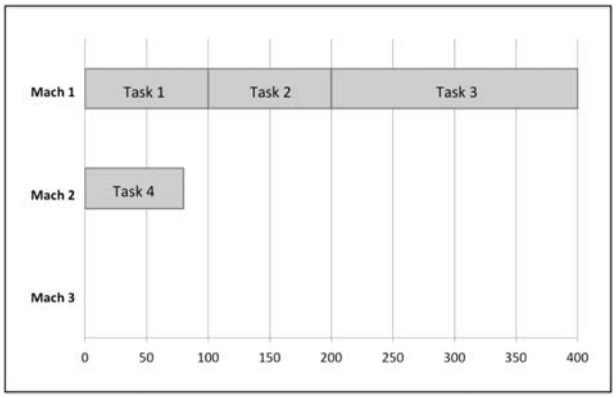

Fig. 1. MET Solution

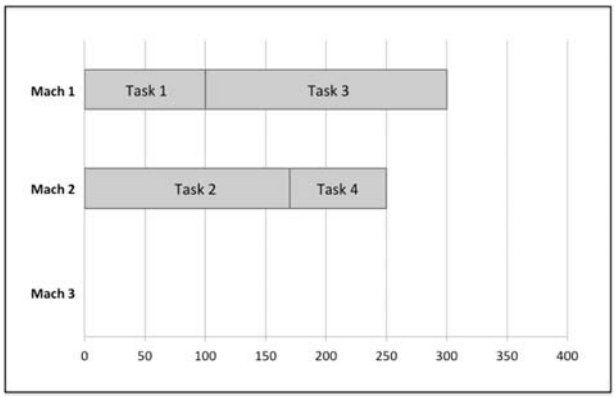

Fig. 2. MCT Solution

3.2.Minimum Completion Time

The *MCT* (*Minimum Completion Time*) heuristic uses the completion time to allocate tasks; this takes into consideration tasks already allocated, in opposition to *MET* heuristics (Braun et al., 2001; Briceño et al., 2012; Ibbara et al., 1997). The *MCT* algorithm is:

1. Unmapped tasks are placed into a list in a random sequence;
2. The first is allocated to the minimum completion time machine;
3. The task selected in step 2 is removed from the tasks list;
4. Ready times are updated with assigned task execution time;
5. Steps II, III and IV are repeated until all tasks have been mapped.

In the illustrative example, *MCT* found a solution (Fig. 2) with *makespan* of 300 *t.u*, with the allocation of tasks 1 and 3 to machine 1 and tasks 2 and 4 to machine 2.

3.2.Min-Min

The *Min-Min* heuristic follows a two-phased approach. First, the machines with the minimum completion time for each task are determined, then, between the task/machine pairs determined in the first step the one with lowest completion time is allocated. This approach makes this heuristic indifferent to the sequence in which the tasks are allocated (Braun et al., 2001; Briceño et al., 2012; Chaturvedi & Sahu 2011; Gupta & Singh, 2012) The *Min-Min* algorithm is:

1. Unmapped tasks are placed into a list in a random sequence;
2. For each listed task, the machine that provides the minimum completion time is identified;
3. For each task/machine pair found in step 2, the one with the minimum completion time is identified;
4. The task determined in step 3 is removed from the task list and allocated to the selected machine;

5. Machine ready times are updated with the assigned task execution time;
6. Steps 2, 3, 4 and 5 are repeated until all tasks have been mapped

In the illustrative example depicted in Fig. 3, *Min-Min* obtained a solution with *makespan* of 380 *t.u.*, with the allocation of tasks 1 and 2 into machine 1 and tasks 3 and 4 allocated into machine 2. The obtained solution is far from the optimal solution that would have a *makespan* of 260 *t.u.* There are several other heuristics that use a similar two-phase approach, such as *Max-Max* or *Max-Min* heuristics.

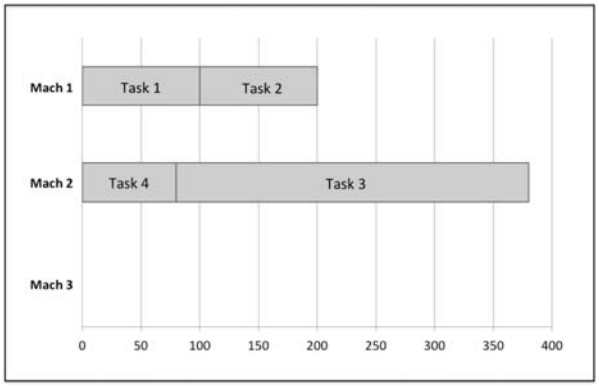

Fig. 3. Min-Min Solution

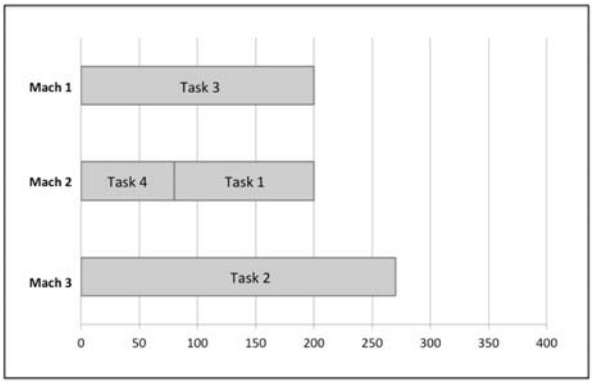

Fig. 4. Suffrage Solution

3.3.Suffrage

Suffrage finds the minimum and second minimum completion time for each task. When there are several tasks with the minimum completion time in the same machine, the one that would suffer more by being allocated to the machine with the second best completion time is allocated to the minimal completion time machine (Briceño et al., 2012; Gupta & Singh, 2012; Paranhos & Brasileiro 2003). *Suffrage* allocates first the tasks that would have more impact in the *makespan*, when they are not allocated to their favorite machine. The *Suffrage* algorithm is:

1. Unmapped tasks are placed into a list in a random sequence;
2. While there are still unmapped tasks;
   i. For each machine find the tasks that have the minimum completion time on that machine;
      a. If there is only one task competing for that machine, allocate it to that machine and remove it from the tasks list;
      b. If there is more than one competing task, allocate the one with the highest Suffrage value and remove it from the task list;
   ii. Machine ready times are updated with the assigned task execution time

The suffrage value can be calculated as seen in Eq. (11), where $2^{nd}MCTM$ represents the second smallest completion time and $1^{st}MCTM$ the smallest completion time machine.

$$S = 2^{nd}\,MCTM - 1^{st}\,MCTM \tag{11}$$

In the illustrative example seen in Fig. 4, *Suffrage* obtained a solution with *makespan* of 270 *t.u*, in three iterations. Task 3 is allocated to machine 1, task 1 and 4 to machine 2 and task 2 to machine 3.

4. Modified Ordered Minimum Completion Time

The *MCT* heuristic, performance depends on the random like order in which tasks are allocated. In this paper, an ordered approach to the *MCT* heuristic is proposed. Santos et al. (2014) proposed an ordered

approach to *MCT*. The *Ordered Minimum Completion Time* (*OMCT*) will determine a sequence of allocation before using the *MCT* algorithm to map tasks. The sequence of allocation will be determined by the standard deviation of the completion times of all tasks and the Suffrage value. The weighted sum of the standard deviation of the completion time on each machine and the Suffrage index will allocate first tasks with the biggest dispersion of their completion time while also taking into consideration the fact that each task is more likely to be allocated to its preferred machines. In the computational study, the *OMCT* heuristic performed better than *MCT*.

In this paper, a *Modified Ordered Minimum Completion Time* (*MOMCT*) is proposed. In this ordered approach to *MCT*, tasks are sequenced in non-increasing order of the MS (Mean Suffrage) value. MS is the difference between the mean of the completion time of a variable number of machines (2º minimum completion time machine, 3º minimum completion time machine, ..., mº minimum completion time machine) and the minimum completion time machine. This does not assume that the tasks will be allocated to their preferred machine. The number of machines considered for the calculation of MS value should be determined, experimentally. The number of machines use in MS should take in consideration how likely are tasks to be allocated into their favorite machines. Several factors can contribute to this, the number of machines (m), the number of tasks (n), the dispersion execution time of tasks between machines (p_j) and the dispersion of execution time of machines between tasks (p_i). The *MOMCT* algorithm is described as follows:

1. Calculate the MS value for each task;
2. Tasks are placed into a list in a decreasing order of their SM value;
3. The first task is allocated to the minimum completion time machine;
4. The task selected in step 3 is removed from the tasks list;
5. Machine ready times are updated with the assigned task execution time;
6. Steps 3, 4 and 5 are repeated until all tasks have been mapped.

The MS can be calculated as seen in Eq. (12), where a is determined experimentally, between the values of [*2; m*]. $i^{nd}MCTM$ represent the completion time on the i^{nd} minimum completion time machine and $1^{st}MCTM$ the completion time on the minimum completion time machine.

$$MS = \frac{\sum_{i=2}^{\alpha} iMCTM}{\alpha - 1} - 1^{st}\,MCTM \qquad \alpha \in [2;m] \tag{12}$$

In the illustrative example of the *MOMCT* heuristic (Fig. 8), with a=3, task 3 is allocated to machine 1, task 2 and task 4 are allocated to machine 2 and task 1 is allocated to machine 3. *MOMCT* heuristic obtained the optimal solution with a *makespan* of 260 *t.u*.

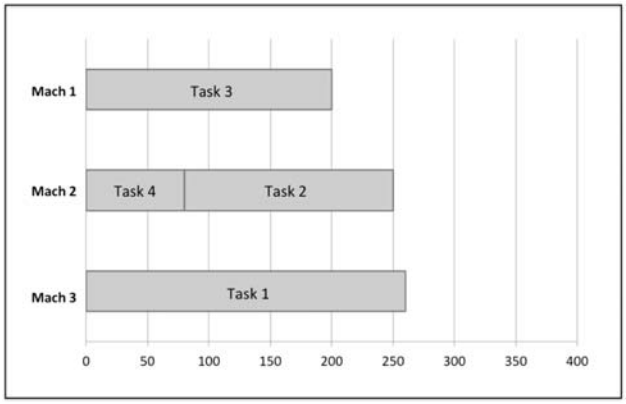

Fig. 5. MOMCT Solution

5. Computational Study

The MCT and MOMCT heuristics were implemented in C in Microsoft Visual Studio 2012 to be compared and evaluated. The computational tests were performed on a MacBook Air with an Intel® Core™ 2 @ 1.60 GHz processors, 4 GB of RAM memory and with Windows 7 64-bit as the operative system with all the available updates. Since the academic benchmark problems as it has been used by multiple authors and in diverse areas, it was considered an effective evaluation method. Both heuristics were tested on benchmark instances. *MOMCT* and *MCT* heuristics performance was evaluated with three groups of 20 instances with variable problem size (No. of Machines X No. of Tasks: 2X5,2X6, 2X7, 2X8, 2X9, 3X5, 3X6, 3X7, 3X8, 3X9, 4X5, 4X6, 4X7, 4X8, 4X9, 5X5, 5X6, 5X7, 5X8 and 5X9) retrieved from Sivasankaran et al. (2010).

5.1.MCT and MOMCT

Both heuristics were compared on their deviation from the optimal *makespan*, since this allows to overview the performance of each heuristic across all problems. Since there are 60 problems, the deviation from the optimal *makespan* can be assumed to follow a normal distribution, according to Central Limit Theorem (*CLT*).

In the computational study *MOMCT* heuristic performed better than the *MCT* heuristic. The bar chart from Fig. 6 depicts the deviation of heuristic obtained solutions from the optimal solution. *MOMCT* found the optimal solution in 30 (50%) instances while *MCT* only found the optimal solution in 21 (35%) instances.

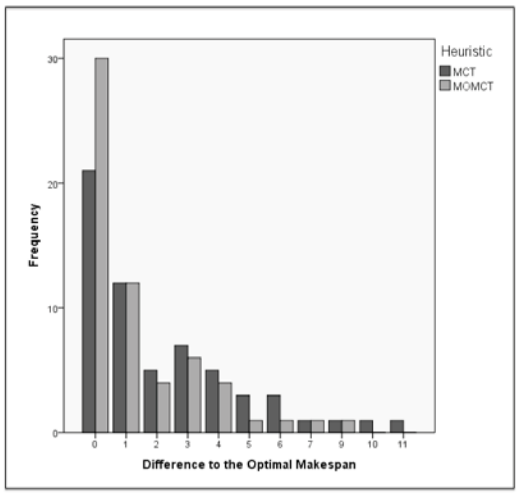

Fig. 6. Bar chart of the solutions of *MCT* and *MOMCT*

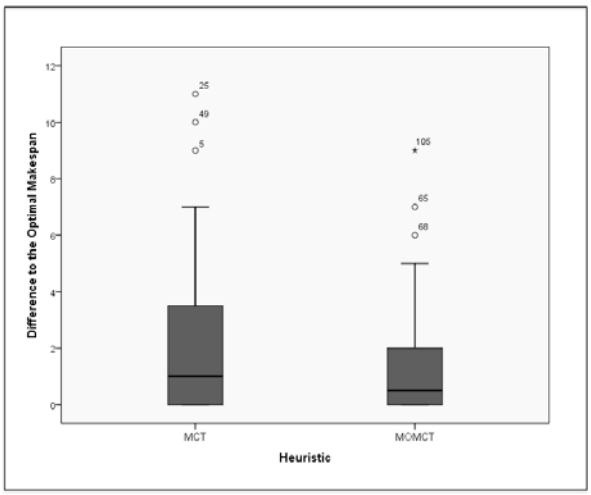

Fig. 7. Boxplot of the solutions of *MCT* and *MOMCT*

The boxplot from Fig. 7 permits to conclude about the advantage of *MOMCT* heuristic, in terms of minimization of *makespan* in the resolution of the analyzed instances. This boxplot allows the analysis of location, dispersion, median and asymmetry of data of the *MOMCT* and *MCT* obtained solutions when compared to the optimal solution of the problems.

To evaluate the significance of the obtained results the Student's t-test for independent samples has been used, considering the hypotheses:

H_0: $\mu_{MCT} - \mu_{OMCT} \geq 0$

H_1: $\mu_{MCT} - \mu_{OMCT} < 0$

Table 2 shows the result of the Student's t-test. It is not possible to assume equal variances (p-value of 0.030) and the student's t-test has a p-value of 0.022 (0.044/2). From the Student's t-test, it is possible to conclude that the MOMCT performs better in problems with more machines, with 95% of confidence level ($p=0.005<\alpha$).

Table 2
Student's T Test Results for *MCT* and *MOMCT*

	Levene's Test for Equality of Variances		t-test for Equality of Means				
	F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference
Equal variances assumed	4.828	0.030	2.035	118	0.044	0.867	0.426
Equal variances not assumed			2.035	108.810	**0.044**	0.867	0.426

5.2. Number of Machines

The *MOMCT* heuristics performance appeared to depend on the number of machines. In the computational tests *MOMCT* seemed to perform better in problems with more machines. This is expected since the *MOMCT* heuristic was implemented to sweep all possible *a* values and due to the fact that *MOMCT* does not assume that the tasks will be allocated to their preferred machine taking more machines in consideration when ordering the tasks. The performance of *MOMCT* with 2/3 and 4/5 machines, where compared on their deviation from the makespan of the optimal solution.

In the computational study *MOMCT* heuristic performed better with more machines (4 and 5). The bar chart from Fig. 8 depicts the deviation of the heuristic obtained solutions from the optimal solution, with 2/3 and 4/5 machines. With 4/5 machines *MOMCT* found the optimal solution in 17 (56.7%) instances while with 2/3 machines it only found the optimal solution in 13 (43.3%) instances.

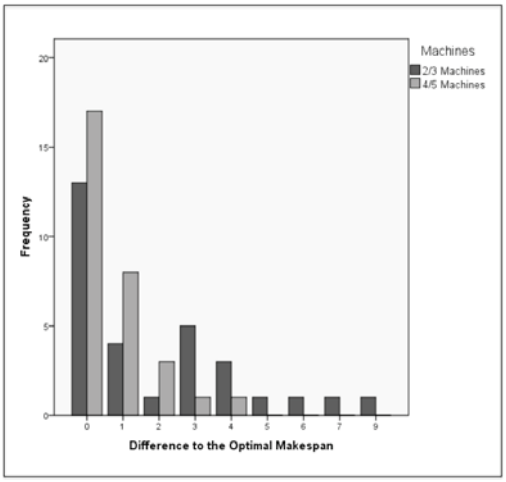

Fig. 8. Bar chart of the solutions for 2/3 and 4/5 machines

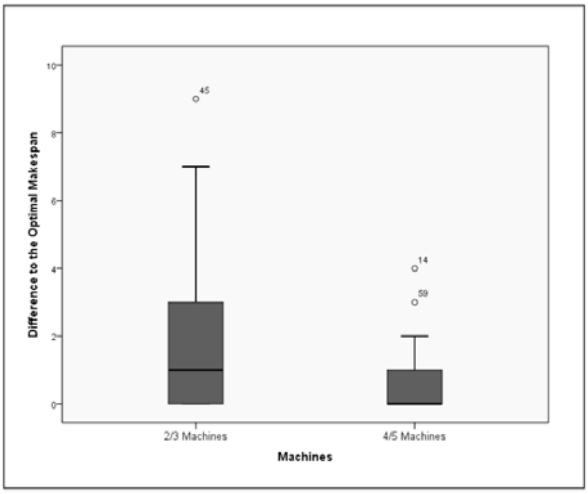

Fig. 9. Boxplot of the solutions for 2/3 and 4/5 machines

The boxplot from Fig. 9 permits to conclude that the *MOMCT* heuristic performs better with 4/5 machines than with 2/3 machines, in terms of minimization of makespan. The boxplot allows the analysis of location, dispersion, median and asymmetry of *MOMCT* with a variable number of machines.

To evaluate the significance of the obtained results the Student's t-test for independent samples has been used, considering the hypotheses:

H_0: $\mu_{2/3} - \mu_{4/5} \geq 0$

H_1: $\mu_{2/3} - \mu_{4/5} < 0$

Table 3 shows the result of the Student's t-test. It is not possible to assume equal variances (p-value of 0.000) and the Student's t-test has a *p-value* of 0.005 (0.010/2). The null hypothesis H_0, that considers the non-existence of difference between the performances on 4/5 and 2/3 machines was rejected ($p=0.005<\alpha$). This permits to conclude, with 95% of confidence that the *MOMCT* heuristic performs better in problems with more machines.

Table 3
Student's T Test Results for 2/3 and 4/5 machines

	Levene's Test for Equality of Variances		t-test for Equality of Means				
	F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference
Equal variances assumed	19.975	0.000	-2.696	58	0.009	-1.300	0.482
Equal variances not assumed			-2.696	38.910	**0.01**	-1.300	0.482

6. Conclusions

In this paper, the *makespan* minimization problem in unrelated parallel-machines was analyzed. Some of the heuristics used to solve the problem were described and compared and a new heuristic was proposed. *MCT* performance depends on a random like order in which tasks are allocated. The possibility of ordering tasks before applying the *MCT* was studied in the proposed heuristic.

The *MOMCT* heuristic orders tasks in accordance to the difference between the mean of the completion time of a variable number of machines and the completion time on the minimum completion time machine. This does not assume that the tasks would be allocated to their preferred machine as several machines can be used to determine in which order tasks would be allocated to machines. Experimental analysis was performed in order to validate the performance of the proposed *MOMCT* heuristic. From the obtained results, it is possible to conclude that *MOMCT* found better solutions than *MCT* with a deviation of 3.00%. *MOMCT* found the optimal solution in 30 (50%) instances while *MCT* only found the optimal solution in 21 (35%) instances. From the obtained results it was possible to conclude with statistical evidence, through the Student's t-test, that the *MOMCT* was more effective than the *MCT* heuristic, with a 95% confidence level. The proposed heuristic performance seems to be dependent on the number of machines in the problem. In the computational study *MOMCT* found the optimal solution in 17 (56.7%) for 4/5 machines and 13 (43.3%) for 2/3 machines. From the Student's t-test it is possible to conclude that the *MOMCT* performs better in problems with more machines, with 95% of confidence level ($p=0.005<\alpha$). This is expected since the *MOMCT* does not assume that the tasks will be allocated to their preferred machine and will sweep all possible *a* values to determine the allocation sequence.

Future work should focus on more computational tests that will compare the performance of the proposed heuristic with other heuristics described in the literature. Moreover, additional emphasize will be given

to compare the performance of the proposed heuristic when dealing with problems of different sizes and the formulation of how the performance of *MOMCT* depends on the number of machines used to order tasks in problems with different characteristics will also be further explored.

Acknowledgements

This work is supported by FEDER Funds through the "Programa Operacional Factores de Competitividade - COMPETE" program and by National Funds through FCT "Fundação para a Ciência e a Tecnologia" under the project: FCOMP-01-0124-FEDER-PEst-OE/EEI/UI0760/2011 and PEst-OE/EEI/UI0760/2014.

References

Baker, K. R., & Trietsch, D. (2009). *Principles of Sequencing and Scheduling* (First Edition). Wiley.

Blazewicz, A.J., Ecker, K. H., Pesh, E., Schmidt, G., & Weglarz, J. (2001). *Scheduling Computer and Manufacturing Process* (Second Edition). Springer.

Braun, T. D., Siegel, H. J., Beck, N., Bölöni, L. L., Maheswaran, M., Reuther, A. I., ... & Freund, R. F. (2001). A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed computing*, *61*(6), 810-837.

Briceño, L. D., Siegel, H. J., Maciejewski, A. A., & Oltikar, M. (2012). Characterization of the iterative application of makespan heuristics on non-makespan machines in a heterogeneous parallel and distributed environment. *The Journal of Supercomputing*, *62*(1), 461-485.

Ebenlendr, T., Krčál, M., & Sgall, J. (2014). Graph balancing: A special case of scheduling unrelated parallel machines. *Algorithmica*, *68*(1), 62-80.

Chaturvedi, A. K., & Sahu, R. (2011). New heuristic for scheduling of independent tasks in computational grid. *International Journal of Grid and Distributed Computing*, 4(3), 25-36.

Fujimoto, N., & Hagihara, K. (2004). A Comparison among grid scheduling algorithms for independent coarse-grained tasks. *SAINT Workshops*, 674-680.

Glass, C. A., Potts, C. N., & Shade, P. (1994). Unrelated parallel machine scheduling using local search. *mathematical. Computer Modeling*, 20(2), 41-52.

Gupta, K., & Singh, M. (2012). Heuristic based task scheduling in grid. *International Journal of Engineering and Technology*, 4(4), 254-260.

Hariri, A., & Potts, C. (1991). Heuristics for scheduling unrelated parallel-machines. *Computers and Operations Research*, 18(3), 323–331.

Ibbara, O., & Kim, C. (1977). Heuristic algorithms for scheduling independent tasks of no identical processors. *Journal of Association for Computing Machinery*, 24(2), 280-289.

Lenstra, J. K., Shmoys, D. B., & Tardos, E. (1990). Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46(1-3), 256-271.

Lin, Y. K., Pfund, M. E., & Flower, J. W. (2011). Heuristics for minimizing regular performance measures in unrelated parallel machine scheduling problems. *Computer & Operations Research*, 38(6), 901-916.

Martello, S., Soumis, F., & Toth, P. (1997). Exact and approximation algorithms for makespan minimization on unrelated parallel machines. *Discrete Applied Mathematics*, 75(2), 169-188.

Paranhos, D., Cirne, W., & Brasileiro, F. V. (2003). Trading cycles for information: Using replication to schedule bag-of-tasks application on computational grids. *International Conference on Parallel and Distributed Computing*, 2790, 169-180.

Pfund, M., Flower, J. W., Jatinder, & Gupta, J. N. D. (2004). A survey of algorithms for single and multi-objective unrelated parallel-machine deterministic scheduling problems. *Journal of the Chinese Institute of Industrial Engineers*, 21(3), 230-241.

Pinedo, M. L. (2012). *Scheduling Theory, Algorithms, and Systems* (Fourth Edition). Springer.

Serra e Santos, & A., Madureira, A. M. (2014). Ordered minimum completion time heuristic for unrelated parallel-machines. Proceedings of the *9th Iberian Conference on Information Systems and Technologies* (CISTI*2014*), 695-700.

Sivasankaran, P., Sornakumar, T., & Panneerselvam, R. (2010). Efficient heuristic to minimize makespan in single machine scheduling problem with unrelated parallel machines. *Intelligent Information Management*, 2, 188-198.

Sugavanam, P., Siegel, H. J., Maciejewski, A. A., Oltikar, M., Mehta, A., Pichel, R., ... & Pippin, A. (2007). Robust static allocation of resources for independent tasks under makespan and dollar cost constraints. *Journal of Parallel and Distributed Computing*, *67*(4), 400-416.

Vakhania, N., Hernandez, J. A., & Werner, F. (2014). Scheduling unrelated machines with two types of jobs. *International Journal of Production Research*, 52(13), 3793-3801.

Varela, L. R., & Ribeiro, R. A., (2003). Evaluation of simulated annealing to solve fuzzy optimization problems, *Journal of Intelligent & Fuzzy Systems*, 14(2), 59-71.

Varela, M.L.R., Aparício, J.N., & Silva, S.C. (2003), A web-based application for manufacturing scheduling. Proceedings of the *International Conference on Intelligent Systems and Control* (*IASTED*), 400- 405.

Varela, M. L. R., & Carmo-Silva, S. (2008). An ontology for a model of manufacturing scheduling problems to be solved on the web. *Innovation in Manufacturing Networks, 8th IFIP International Conference on Information Technology for Balanced Automation Systems*. Azevedo, A. (Ed.); Springer, 197-204.

Varela, M. L. R., Barbosa, R., & Putnik, Goran (2012). Experimental platform for collaborative inter and intra cellular fuzzy scheduling in an ubiquitous manufacturing system, Proceedings of the *First International Conference on Virtual and Network Organizations Emergent Technologies and Tools* (*ViNOrg'11*), 227-236.

Varela, M. L. R., Putnik G. D., & Cruz-Cunha M. M., (2012) Web-based technologies integration for distributed manufacturing scheduling in a virtual enterprise. *International Journal of Web Portals*, 4(2), 19-39.

Varela, M. L. R., Putnik, G. D, & Ribeiro, R. A. (2012). A web-based platform for collaborative manufacturing scheduling in a virtual enterprise. *Information and Communication Technologies for the Advanced Enterprise an international journal*, 2, 87-108.

Varela, M. L. R., & Ribeiro, R. A. (2014), Distributed manufacturing scheduling based on a dynamic multi-criteria decision model. *Studies in Fuzziness and Soft Computing*, 317, 81-93.

Verschae, J., & Wiese, A. (2014). On the configuration-LP for scheduling on unrelated machines. *Journal of Scheduling*, 17, 371-383.

Xhafa, F., & Abraham, A. (2008). *Metaheuristics for scheduling in industrial and manufacturing applications series: Studies in computational intelligence,* 128, Springer.