Fanjul Peyró, L.; Ruiz García, R. (2011). Size-reduction heuristics for the unrelated parallel machines scheduling problem. Computers and Operations Research. 38(1):301-309. doi:10.1016/j.cor.2010.05.005.

Size-reduction heuristics for the unrelated parallel machines scheduling problem

Luis Fanjul-Peyro, Rubén Ruiz*

Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática,

Ciudad Politécnica de la Innovación, Edificio 8G. Acceso B. Universidad Politécnica de Valencia,

Camino de Vera s/n, 46022 Valencia, Spain

lfpeyro@terra.es, rruiz@eio.upv.es

May 6, 2010

Abstract

In this paper we study the unrelated parallel machines problem where n independent jobs must be assigned to one out of m parallel machines and the processing time of each job differs from machine to machine. We deal with the objective of the minimisation of the maximum completion time of the jobs, usually referred to as makespan or $C_{\max}$. This is a type of assignment problem that has been frequently studied in the scientific literature due to its many potential applications. We propose a set of metaheuristics based on a size-reduction of the original assignment problem that produce solutions of very good quality in a short amount of time. The underlying idea is to consider only a few of the best possible machine assignments for the jobs and not all of them. The results are simple, yet powerful methods. We test the proposed algorithms with a large benchmark of instances and compare them with current state-of-the-art methods. In most cases, the proposed size-reduction algorithms produce results that are statistically proven to be better by a significant margin.

Keywords: Unrelated parallel machines, Makespan, Size-reduction

*Corresponding author. Tel: +34 96 387 70 07, ext: 74946. Fax: +34 96 387 74 99

1 Introduction

In the unrelated parallel machines scheduling problem the aim is to schedule a set N of n jobs on a set M of m machines that are disposed in parallel, this is, all machines are capable of processing all jobs. Each job has to be processed by exactly one machine. Each machine cannot process more than one job at the same time and when a job begins its processing it cannot be stopped until its completion, i.e., preemption is not allowed. The processing times of jobs $j = 1, \ldots, n$ in machines $i = 1, \ldots, m$ are denoted by p_{ij}. These processing times are non-negative, fixed and known in advance. There are some variants of this problem, all of them are special cases. For example, in the identical parallel machines scheduling problem, each job has the same processing time p_j no matter the machine to which it is assigned to. There is also the case of uniform parallel machines, where the processing time of a job j on machine i follows the relationship $p_{ij} = p_j/s_i$, where s_i denotes a different speed for machine i when processing the jobs, i.e., according to s_i, some machines are "faster" or "slower". As we can see, the unrelated parallel machines case is the most general as the processing times depend both on the machine and on the job. This paper deals with this last and most general case. The optimisation criterion is the minimisation of the maximum completion time or makespan (C_{max}). Thus the studied scheduling problem in this paper is denoted by $R//C_{\max}$, according with the $\alpha/\beta/\gamma$ classification scheme of Graham et al. (1979). The $R//C_{\max}$ problem is, in fact, a type of assignment problem. Let us denote by J_i the set of jobs that have been assigned to machine i. Machine i will be busy during $C_i = \sum_{\forall k \in J_i} p_{ik}$ units of time where C_i is the completion time of machine i. As one can easily see, the order in which these J_i jobs are processed does not influence the value of C_i. Following these definitions, $C_{\max} = \max_{i \in M}\{C_i\}$. For the problem $R//C_{\max}$, the total number of possible solutions is m^n. $R//C_{\max}$ is an $\mathcal{NP}$-Hard problem in the strong sense, since Garey and Johnson (1979) demonstrated that, in the special case with identical machines (referred to as $P//C_{\max}$), the problem is already $\mathcal{NP}$-Hard. Furthermore, Lenstra et al. (1977) showed that even the two machines version ($P2//C_{\max}$) belongs to that complexity class. An straightforward Mixed Integer Linear Programming (MILP) formulation for the $R//C_{\max}$ is as follows:

$$\min \ C_{\max} \tag{1}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \forall j \in N \tag{2}$$

$$\sum_{j=1}^{n} p_{ij} \cdot x_{ij} \leq C_{\max} \quad \forall i \in M \tag{3}$$

$$x_{ij} \in \{0, 1\} \quad \forall j \in N, \forall i \in M \tag{4}$$

Where x_{ij} is a binary variable which takes value 1 if job j is assigned to machine i and 0 otherwise.

There are many examples and real problems that can be modeled as a $R//C_{\max}$, specially production systems where one or more production stages have several machines in parallel so to increase productivity. Modern multi-core processors can also be seen as parallel machines problems. Examples and general literature is available from Pinedo (2005, 2008) and Sule (2008), among many others. Some of the earlier works on parallel machines are due to McNaughton (1959) and since then, many papers about parallel machines have appeared, as was shown in the reviews of Cheng and Sin (1990) and Mokotoff (2001). A recent short review, more focused in the unrelated parallel machines case is provided by Fanjul-Peyro and Ruiz (2010). Here, a brief summary of the most important related work is commented. Note that we try to refer to the $R//C_{\max}$ problem only, as the literature on all versions of parallel machines problems is remarkably extensive. Horowitz and Sahni (1976) proposed a Dynamic Programming approach and some heuristic methods for the two unrelated parallel machines and makespan criterion. Ibarra and Kim (1977) presented five heuristic algorithms and a tight error bound result for the LPT dispatching rule. De and Morton (1980), Davis and Jaffe (1981) and Lenstra et al. (1990) introduced some heuristic and approximation algorithms. Contrary to other scheduling problems, where exact approaches are valid only for special cases and/or instances of limited size, the exact results for the $R//C_{\max}$ in the literature are impressive. Martello et al. (1997) presented a Branch and Bound (B&B) method improving the results provided by van de Velde (1993) that was able to solve to almost optimality instances of up to 80 jobs and 20 machines (80×20).
There is a large number of papers in the literature that exploit the idea of solving a linear relaxation of the MILP model in a first phase and using a rounding method, that can be exact or approximate, in a second phase in order to obtain a feasible solution to the problem. This technique was first employed by Potts (1985), followed by Lenstra et al. (1990), Shmoys and Tardos (1993) and Shchepin and Vakhania (2005), among others. Hariri and Potts (1991) were among the first to integrate local search with heuristics. van de Velde (1993) proposed an exact algorithm and an iterated local search metaheuristic. Glass et al. (1994) proposed Genetic Algorithm (GA), Tabu Search (TS) and Simulated Annealing (SA) methods. Piersma and van Dijk (1996) showed a SA and a TS with initialisations coming from the aforementioned heuristics of Hariri and Potts (1991) and Davis and Jaffe (1981). Srivastava (1998) presented another TS and Sourd (2001) published two methods based on large neighbourhood search. A heuristic cutting planes method was developed by Mokotoff and Chretienne (2002) that was later refined and extended in what is known as the "Partial" method in Mokotoff and Jimeno (2002). This algorithm is based on the methodology of Dillenberger et al. (1994) which was later used by Mansini and Speranza (1999). Woclaw (2006) carried out a comprehensive re-implementation and a computational evaluation of most existing literature for the $R//C_{\max}$, and showed that the Partial method of

Mokotoff and Jimeno (2002) produced state-of-the-art results, surpassing all other tested methods. Lately, Ghirardi and Potts (2005) have shown very good results using a Recovering Beam Search (RBS) approach which was already studied earlier by de la Croce et al. (2004), based on the Beam Search (BS) method of Ow and Morton (1988). Gairing et al. (2007) also proposed a two-phase approach that results in a very fast method but with results that are not able to compete in solution quality with those of Mokotoff and Jimeno (2002) or Ghirardi and Potts (2005). Recently, Fanjul-Peyro and Ruiz (2010) introduced a number of simple local search methods, based on the *Iterated Greedy* (IG) method of Ruiz and Stützle (2007) and on fast local search algorithms on nested neighbourhoods, following the Variable Neighbourhood Descent approach (VND) of Mladenovic and Hansen (1997) or Hansen and Mladenovic (2001). These methods were shown to clearly outperform the Partial and RBS methods of Mokotoff and Jimeno (2002) and Ghirardi and Potts (2005) in comprehensive benchmarks. Furthermore, Fanjul-Peyro and Ruiz (2010) also demonstrated that modern commercial solvers, and more specifically, IBM-ILOG CPLEX version 11.0 (simply referred to as CPLEX from now on, although we will make special mention of different version numbers) are capable of outperforming the Partial of Mokotoff and Jimeno (2002) and of producing results that are statistically equivalent to the RBS of Ghirardi and Potts (2005). Note that this good performance of CPLEX was tested with "out-of-the-box" parameters and with the previous simplistic assignment model. According to this last work of Fanjul-Peyro and Ruiz (2010), the new state-of-the-art is formed by CPLEX and by their best performing proposed method, referred to as NVST-IG+ in their original paper.

One of the advantages of NVST-IG+ is that, contrary to the well known two-phase approaches, including the Partial algorithm of Mokotoff and Jimeno (2002), no commercial solver is necessary as NVST-IG+ is a stand alone local search method. However, we postulate that given the extremely good performance of CPLEX, even better solutions can be obtained by combining NVST-IG+ with a solver. In this work we challenge the simple and high performing methods of Fanjul-Peyro and Ruiz (2010) and propose new metaheuristics that make extensive use of solvers. However, our approach is radically different from the well-known two-phase approaches. Instead of solving a linear relaxation of the MILP model and then rounding the results to reach a feasible solution, we solve a reduced version of the MILP model where only a few of the best possible machine assignments for the jobs are considered, i.e., we drastically reduce the number of x_{ij} binary variables. The resulting sub-optimal solution is then improved by local search methods. As we will show, this approach produces simple and very well performing methods.

This rest of the paper is organised as follows: Section 2 details the algorithms proposed. Section 3 details the experimental settings to be used in the experimentation. Extensive computational and statistical analyses are presented in Section 4. Finally, some concluding remarks and future research directions are given in Section 5.

2 Proposed methods

Fanjul-Peyro and Ruiz (2010) employed a total of 1400 instances divided into 7 groups with different distributions of the processing times p_{ij} with 200 instances each, ranging from the smallest instances of 100 jobs and 10 machines to instances of up to 1000 jobs and 50 machines. Fanjul-Peyro and Ruiz (2010) used a modern solver to solve this large benchmark of instances, something that was not done in the literature before. These 1400 instances were solved with a CPU time limit of 2 hours with IBM-ILOG CPLEX version 11.0 on a PC/AT computer with a Pentium IV processor running at 3.0 GHz and 1 GB of RAM memory. About 34% of instances could be solved to optimality. While this is not a large number, for the unsolved instances, the average gap was of around just 1%. In any case, we are referring to results after two lengthy hours of CPU time. However, for some instance groups and sizes, the maximum gaps reached almost 9%. Still, it was surprising to find that, in some cases, CPLEX could solve 100% of instances of up to 200×50. CPLEX was even able to solve many instances of 1000×50 (this is, 50,000 binary variables) in under two hours of CPU time. This unexpected excellent behaviour of CPLEX posed it as a true challenger against the established state-of-the-art. Basically, CPLEX as shown in Fanjul-Peyro and Ruiz (2010) to be competitive, or even better than the best available proposed methods from the literature, but not better than their best proposed method. Furthermore, a close observation indicated that in many occasions, CPLEX was either trying to reduce an already very small gap or was proving the optimality of a solution. A close analysis of the resulting solutions yielded an expected result: Most jobs were assigned to their fastest machine. This was predictable as a result of the assignment problem. However, it was surprising how overwhelmingly consistent this result was. In almost all situations, each job was assigned to either the fastest, second fastest or third fastest machine. In other words, among the $n \cdot m$ binary variables of the previous MILP model, jobs were assigned to machines i_{1j}, i_{2j} or i_{3j} where $i_{1j} = min_{i \in M}\ p_{ij}$, $i_{2j} = min_{i \in M/i_{1j}}\ p_{ij}$ and $i_{3j} = min_{i \in M/i_{1j}, i_{2j}}\ p_{ij}$. If jobs are rarely assigned to other slower machines, then a natural question arises: Is it needed to define all other variables? A "reduced" MILP model considering only these "three fastest machines" has $n \cdot 3$ binary variables. This means that if a 1000×50 instance needs 50,000 binary variables, by just using these three fastest machines for each job all we need is 3000 variables, i.e., 16.67 times less variables. Of course, this size reduction comes at a high cost. First of all, optimality cannot be guaranteed and second, this straightforward size reduction is not expected to perform that well in all cases. Let us study in detail this unexpected behaviour of the CPLEX solver. For each one of the previously mentioned 1400 instances and for the reference solutions obtained with CPLEX with a 2 hours termination criterion on each instance, we count how many jobs were not assigned to any of the three fastest machines i_{1j}, i_{2j} or i_{3j}. Results are given in Table 1. Instances are grouped by 20 different combinations of n and m, with 10 instances at each group. Recall also that there are 7 big groups of instances where the difference is how the processing times p_{ij} are distributed. Further details about this benchmark will be given later in this paper. Each cell in Table 1

represents the percentage of jobs in the instance that were assigned to a machine not in i_{1j}, i_{2j} or i_{3j} in the final mentioned reference solution.

n	m	$U(1,100)$	$U(10,100)$	$U(100,200)$	$U(100,120)$	$U(1000,1100)$	jobcorre	machcorre
100	10	0.30	0.50	0.80	1.30	1.10	0.70	30.20
	20	1.40	1.20	4.60	1.50	2.80	4.40	35.40
	30	3.80	4.60	18.00	24.40	31.20	10.90	41.30
	40	3.50	7.30	20.40	11.10	24.70	17.90	43.80
	50	5.50	10.20	6.60	1.50	4.80	32.40	41.30
200	10	0.20	0.10	0.30	0.30	0.50	0.30	31.90
	20	0.40	0.65	1.40	0.55	1.30	2.20	40.30
	30	0.80	1.65	6.80	17.75	10.50	4.75	44.05
	40	1.20	1.85	1.85	0.60	3.20	9.90	45.75
	50	2.15	3.65	4.30	0.25	4.35	12.60	46.15
500	10	0.04	0.00	0.06	0.10	0.08	0.02	32.82
	20	0.08	0.12	0.30	0.20	0.24	0.18	43.14
	30	0.32	0.12	2.34	13.46	8.80	0.80	44.48
	40	0.52	0.28	4.46	11.00	10.12	2.10	44.30
	50	0.36	0.74	0.84	0.28	0.84	4.20	45.10
1000	10	0.02	0.01	0.00	0.02	0.02	0.00	29.66
	20	0.04	0.02	0.19	0.13	0.12	0.06	37.65
	30	0.05	0.09	0.74	1.78	12.57	0.25	39.84
	40	0.13	0.09	0.29	0.16	0.17	0.86	46.67
	50	0.20	0.11	0.45	0.10	0.22	1.33	45.14
Average(%)		1.05	1.66	3.74	4.32	5.88	5.29	40.45

Table 1: Percentage of jobs at each instance that are not assigned to any of the three fastest machines i_{1j}, i_{2j} or i_{3j} in the reference solutions given by IBM-ILOG CPLEX 11.0 with a time limit of 2 hours.

From Table 1 we notice that in the interval where the processing times are randomly uniformly distributed in the interval $[1, 100]$, i.e., $U(1, 100)$, only 1% of jobs are, on average, assigned to machines other than the three fastest. Excluding the last column, no interval exceeds 6%. Take for example the largest instances of 1000×50 and the interval $U(1, 100)$. We see that, on average from the 10 instances, 0.2% jobs are assigned outside the three fastest machines. This means that a meager 2 jobs are assigned to "slower" machines. The last column shows instances where processing times are machine correlated. In this case, jobs are not only assigned to fast machines. In machine correlated instances, there are machines which are faster for all jobs and therefore, in the reference solution not all jobs are expected to be assigned to the fastest machines as they quickly become overloaded.

Table 1 only indicates the percentage of jobs assigned outside the three fastest machines. Given those numbers, it is expected that a size-reduced model considering only the three fastest machines will perform good and a low $C_{\max}$ value is to be hoped for. This is true in most situations. For example, after only 15 seconds of CPU time, the sized-reduced MILP model for one instance in the interval $U(1, 100)$, with size 500×40 gives a 2.5% gap, which happens to be better than the gap obtained with the full MILP model after 2 hours of CPU time. However, for the same interval, we find an extreme example of an instance of size 100×50 that after 15 seconds, the

reduced MILP obtains a solution with 12.5% gap, much worse than the one obtained with the full MILP model and 2 hours of CPU time. To conclude, we have observed this interesting CPLEX behaviour but special care has to be put in how the size-reduced MILP models are obtained.

2.1 Size-reduction methods

In the previous sections we have explained how apparently not all $n \cdot m$ x_{ij} binary variables in the MILP model are needed in order to reach high quality solutions in a much shorter amount of time. We now present some simple approaches that allow to select a reduced subset of machines for each job prior to using a solver. The first approximation is to select the k best (minor) processing times p_{ij} for each job. Note that we will refer to the selected x_{ij} binary variables or p_{ij} values interchangeably. In this case, the $k \cdot n$ smallest p_{ij} values are selected. We denote this method as job-based size reduction or kJB. Table 2 shows an example of $3JB$ for an instance with 10 jobs and 5 machines, where the three minor values of p_{ij} are selected for each job.

	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
M1	5	9	**3**	6	**4**	**3**	6	**5**	7	**3**
M2	8	9	7	6	6	4	**5**	**1**	**1**	4
M3	**1**	**1**	**4**	**3**	8	**3**	7	**4**	8	**1**
M4	**2**	**8**	**3**	**4**	**2**	**3**	**1**	7	**1**	6
M5	**3**	**7**	8	**5**	**2**	8	**4**	9	**5**	**3**

Table 2: Example of selected values of p_{ij} in 3JB size-reduction method. Selected values in bold.

As it was concluded from Table 1, this method selects appropriate values in most cases but it is a bad choice for correlated machines. We need a size-reduction method to select p_{ij} within machines instead of within jobs. First we calculate the theoretical average of jobs that should be assigned to each machine in an ideal solution, $a = n/m$, and use the value of a to determine how many values of p_{ij} to select at each machine. However, some machines could have more jobs assigned than the theoretical average. To counter this, we select $l \cdot a$ values of p_{ij} at each machine, and the total number of p_{ij} selected values with this method is therefore $l \cdot a \cdot m$. We denote this method as machine-based size reduction or lMB in short. An example of $2MB$ can be observed in Table 3. Notice that for this example $a = 10/5 = 2$ and $l = 2$ so actually, four values are selected at each row of the table.

	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
M1	5	9	**3**	6	**4**	**3**	6	5	7	**3**
M2	8	9	7	6	6	***4***	5	**1**	**1**	**4**
M3	**1**	**1**	4	3	8	**3**	7	4	8	**1**
M4	**2**	8	3	4	**2**	3	**1**	7	**1**	6
M5	**3**	7	8	5	**2**	8	**4**	9	5	**3**

Table 3: Example of selected values of p_{ij} in 2MB size-reduction method. Selected values in bold.

There is an important drawback in the lMB size-reduction method. Since we are selecting only the minor p_{ij} values across machines (rows in Table 3) some job might be left without any possible p_{ij} value. This would lead to an infeasible solution in the size-reduced MILP model. Actually, this is the case for job J4 in Table 3. The solution for this problem is to use the two previous size-reduction methods together, i.e., kJB and lMB. Obviously, much of the selected p_{ij} values of both methods will be the same, but not all. For example, in Table 3 we can notice that the selected value $p_{26} = 4$ it is not selected in Table 2.

The previous size-reduction methods are simple and, as we will see, allow for good results. However, the literature is full of very efficient and effective methods for the $R//C_{\max}$ problem and it seems plausible to select the minor p_{ij} values for the size-reduced MILP after observing the solutions given by effective heuristics. For this purpose, we select the best known heuristic for this problem, the already mentioned NVST-IG+ algorithm of Fanjul-Peyro and Ruiz (2010). In what follows, we will simply refer to this heuristic as DIG in short (variable neighbourhood Descent approach with Iterated Greedy). Basically, DIG is run for a very short amount of time (5 seconds) and each time a new best solution is found by DIG during these 5 seconds, the p_{ij} values of such solution are selected without repetition. An example of this method is given in Table 4 where a solution obtained by DIG in a iteration is shown.

	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
M1	***5***	9	3	6	4	3	6	5	7	3
M2	8	9	7	6	6	**4**	5	**1**	1	4
M3	1	**1**	4	**3**	8	3	7	4	8	1
M4	2	8	**3**	4	2	3	**1**	7	**1**	6
M5	3	7	8	5	**2**	8	4	9	5	**3**

Table 4: Example of selected values of p_{ij} in DIG size-reduction method. Selected values in bold.

It is also possible to mix DIG with the other size-reduction methods, like for example kJB and DIG to obtain a hybrid method with all p_{ij} values of kJB method plus all p_{ij} different values

provided by DIG. Following the previous examples, we can notice that the $p_{11} = 5$ value of DIG method from Table 4 is not present in the selected values of the $3JB$ method from Table 2.

2.2 Size-reduction algorithms

The previous size-reduction methods only simplify the original MILP model but do not constitute solution methods in themselves. However, pairing these size-reduction methods with CPLEX already gives us a complete solution. Notice the simplicity of the proposed methods in two phases 1) Size-reduce the original p_{ij} or x_{ij} matrix and 2) run CPLEX over this size-reduced problem until a pre-specified CPU time termination criterion is met. With this approach we propose three size-reduction algorithms as follows:

1. kJB size-reduction method, with $k = 3$ and then run CPLEX. We simply refer to this size-reduction algorithm as 3J.

2. Combined size-reduction methods kJB and lMB with $k = 3$ and $l = 2$ and then run CPLEX. This second size-reduction algorithm is referred to as 3J2M.

3. Combined size-reduction method kJB, with $k = 3$, together with the size-reduction method DIG. We name this algorithm 3JD. Note that this last method is more complex, as it requires the DIG heuristic (NVST-IG+ algorithm from Fanjul-Peyro and Ruiz, 2010).

Note that the optimum CPLEX solution in the second phase is optimum for the size-reduced problem which generally does not translate into an optimum solution for the original problem. The three previous algorithms can be significantly improved. As CPLEX is run for a size-reduced problem it is very likely that an optimum solution is found much before the pre-specified maximum CPU time termination criterion is reached. The most logical approach is to iterate the size-reduction algorithm with more selected p_{ij} values, i.e., repeat the algorithm with a bigger x_{ij} matrix and keep doing so while there is still time available. Quite differently is the case when CPLEX spends lots of time closing the search tree, proving optimality but without actually improving the current solution. In these circumstances, it makes little sense to waste valuable CPU time. There is a possibility, for example, to stop the solver in the second phase after a pre-specified elapsed CPU time and repeat the size-reduction algorithm with a bigger x_{ij} matrix. With all this in mind, two additional size-reduction algorithms are proposed:

4. 2JDi size-reduction algorithm: It uses the kJB size-reduction method starting with $k = 2$, together with the size-reduction method DIG, and then run CPLEX. Each time CPLEX ends and the stopping criterion is not met, k is increased by 1, the new p_{ij} values are added and CPLEX is run again. Note that the DIG size-reduction method is only run once at the first iteration.

5. 2JDi(90) size-reduction algorithm. It is similar to 2JDi with the only difference that each time CPLEX is run, a maximum CPU time limit of 90 seconds is imposed.

Note that the 90 seconds CPU time limit for 2JDi(90), as well as the values of k and l in the previous proposed methods were obtained after calibrations and testings which will be further detailed in later sections.

2.3 Parallel size-reduction algorithms

Nowadays, regular desktop computers have more than one processor integrated in the same chip, these are the well known multiple cores. At the time of the writing of this paper, there are commercially available processors with 6 physical cores inside the same chip. This increased available power is used by modern solvers like CPLEX which runs in parallel mode if several physical cores or processors are available. We also propose some simplistic parallel versions of the previous size-reduction algorithms to use all the power provided by multi-core processors.
The simplest possible parallel approach is to run as many instances of our proposed size-reduction algorithms as cores are available and basically keep the best result at the end, i.e., there are no communications between the different instances. This "brute-force" approach should be seen as the simplest possible multi-core method. Fanjul-Peyro and Ruiz (2010), in their proposed NVST-IG+ algorithm did not propose a parallel version. Here, we test the M-DIG (multicore DIG) which is basically the full DIG heuristic being run multiple times.
For our proposed size-reduction algorithms we propose similarly simple parallel versions. More specifically, we propose the multicore 2JDi(90) or M-2JDi(90). The basic functioning of this parallel version is straightforward. Picture for example a processor with two cores, therefore, M-2JDi(90) is launched in core 1 and M-3JDi(90) is launched in core 2, i.e., each core is working on a different x_{ij} size-reduced matrix. The first core that finishes is assigned the next bigger size-reduced matrix: M-4JDi(90). The process continues until the termination criterion is met. Note that we do not propose parallel versions for 3J, 3J2M, 3JD or 2JDi as M-2JDi(90) easily dominated the rest, as will be shown next.

3 Experimental settings

When demonstrating the performance of the new proposed methods, special care must be given to the experimental settings. We first explain in detail the large benchmark of instances employed and how by just changing the way the processing times are distributed, very different results are obtained. Since we will be testing different solvers and algorithms, both in serial as well as in parallel mode, further details of the computers used will be also given.

3.1 Benchmark of instances

Fanjul-Peyro and Ruiz (2010) showed that there was not a publicly available benchmark of instances for the $R//C_{\max}$ problem and each author used his/her own instances. To remedy that

situation, they proposed a comprehensive benchmark and made it available, along with the reference solutions in `http://soa.iti.es`. For the sake of completeness, this benchmark is now detailed.

In general, an instance of $R//C_{\max}$ is defined by the number of jobs n, number of machines m and the processing times p_{ij} matrix of size $m \cdot n$. It is well known that the distribution of the p_{ij} values have a great influence in the algorithms and on the quality of the solutions obtained. Fanjul-Peyro and Ruiz (2010) proposed seven distributions for the p_{ij} values. Some of them were of common use in the literature and some others were newly proposed. The most common distributions in the literature are $U(1, 100)$ and $U(10, 100)$. Some authors also use instances where the processing times are job or machine correlated. For correlated jobs, the processing times are determined by $p_{ij} = b_j + d_{ij}$ where b_j and d_{ij} are uniformly distributed values in the ranges $U(1, 100)$ and $U(1, 20)$, respectively. In the case correlated machines, the processing times are derived from the expression $p_{ij} = a_i + c_{ij}$ where a_i and c_{ij} are uniformly distributed in $U(1, 100)$ and $U(1, 20)$, respectively. The newly proposed intervals try to model more real instances with the intervals $U(100, 200)$, $U(100, 120)$ and $U(1000, 1100)$. For each one of these seven intervals there are the following combinations of $m = \{10, 20, 30, 40, 50\}$ and $n = \{100, 200, 500, 1000\}$. In total there are 20 combinations, from 100×10 to 1000×50. 10 random different instances are defined for each combination which results in 200 instances in each one of the seven intervals and 1400 instances in total. To the best of our knowledge, no such comprehensive and large benchmark exists for the $R//C_{\max}$ problem.

3.2 Set of computers

In this paper, all tests have been carried out in a cluster of 12 PC/AT computers with Intel Core 2 Duo E6600 processors, running at 2.4GHz with 2 GB of RAM memory under Windows XP SP3 operating system. Each processor contains two physical CPUs (cores). Special mention should be given to IBM-ILOG CPLEX. Fanjul-Peyro and Ruiz (2010) tested the serial version 11.0 on an older set of computers. In this paper we test the serial and parallel variants of version 11.1. on a much modern set of computers. Notice that it is a small version change. However, we will show that these two facts (faster computers and version change) are enough to boost the performance of CPLEX. As regards the CPU time, all of our proposed methods, the existing DIG algorithm from Fanjul-Peyro and Ruiz (2010) as well as CPLEX are stopped after a given elapsed CPU time. We have fixed this time in 15 seconds as a reference because this was the time needed by CPLEX to solve all instances of 100 jobs in the interval $U(1, 100)$. However, we also test all algorithms with 30, 60, 120, 240 and 300 seconds. All methods are coded in Delphi 2007 and compiled with the optimisation flag enabled. All preliminary calibrations of our algorithms were carried out on the interval $U(1, 100)$ and for a subset of instances. Calibrating for each interval would surely result in even a better outcome, but no further calibrations are carried out in order to avoid over-fitting deviations in the results.

4 Computational analysis

The computational analysis is divided between the serial and parallel methods. More specifically, we carry out comprehensive computational and statistical analyses to assess the performance of the different algorithms. We start our study with CPLEX (forced to use only one core), DIG (the best method originally proposed by Fanjul-Peyro and Ruiz, 2010), 3J, 3J2M, 3JD, 2JDi and 2JDi(90) as defined in previous sections. All these algorithms are serial and use only one core in the CPU. Given that all methods are coded in the same programming language (except CPLEX), run on the same computers with the same settings and under the same CPU time, we expect all results to be fully comparable. Afterwards we will present the results of the parallel algorithms represented by M-DIG, M-2JDi(90) and M-CPLEX using all available cores (2). The response variable that we measure is the relative percentage deviation from the reference solution as follows:

$$\text{Relative Percentage Deviation } (RPD) = \frac{C_{\max}(i) - C^*_{\max}(i)}{C^*_{\max}(i)} \cdot 100 \tag{5}$$

Where $C^*_{\max}(i)$ is the aforementioned 2 hour CPLEX run available from `http://soa.iti.es` (many times optimal or with a very small gap) and $C_{\max}(i)$ is the value obtained by a given algorithm and instance i.

After reporting the results, we carry out statistical analyses in order to soundly test the significance of the given averages. We use statistical testing in order to guarantee that the observed differences in the average results are indeed statistically significant. We carry out a Design of Experiments (DOE, Montgomery, 2009) where we study a single factor, i.e., the type of algorithm at seven levels in the serial case and at three levels in the parallel case. The response variable is the relative percentage deviation. We take all previous results where each instance is considered a treatment. In the serial case there are 1400 instances and seven methods, which results in 9800 observations. For the parallel case there are only three algorithms so there are 4200 observations. This number of observations is gathered for each stopping time. We carry out six independent runs of all methods with 15, 30, 60, 120, 240 and 300 seconds stopping times. As a conclusion, the grand number of observations is of 57,600 results for the serial case and of 25,200 in parallel case. The results of the DOE are analyzed by means of the single factor Analysis of Variance (ANOVA) technique. We check the three main hypotheses of the parametric ANOVA: normality, homocedasticity and independence of the residuals. With such a large set of results, the residuals from the ANOVA easily satisfied all three hypotheses.

4.1 Serial algorithms' results

Our first test comprises the state-of-the-art algorithm DIG proposed by Fanjul-Peyro and Ruiz (2010) and the new CPLEX version 11.1. We put along the first simple algorithms 3J and 3J2M.

Table 5 shows the average results of the relative percentage deviation of all intervals together except for correlated machines. Recall that each cell contains the average of 1400 results. The different stopping times are given in the rows. We do not include the correlated machines interval because as it was shown in Table 1, for this interval the 3J strategy was a poor way of size-reducing the problem. Therefore, these initial results should be seen as a first approximation.

Time (sec.)	CPLEX	DIG	3J	3J2M
15	0.59	0.42	0.52	0.48
30	0.41	0.34	0.43	0.37
60	0.26	0.29	0.38	0.32
120	0.15	0.22	0.34	0.28
240	0.09	0.17	0.30	0.24
300	0.07	0.16	0.28	0.22

Table 5: Average Relative Percentage Deviations for all intervals except correlated machines for existing state-of-the-art and simple size-reduction algorithms 3J and 3J2M.

The first remarkable result is that CPLEX has seen its performance increased in a significant way since the results Fanjul-Peyro and Ruiz (2010). This is due to the new version 11.1 and due to the faster computers used. In Fanjul-Peyro and Ruiz (2010) it was stressed that regular commercial solvers should not be overlooked since they steadily improve performance and this is a clear demonstration of this fact. A second remarkable result is that DIG is now competitive with CPLEX for very short CPU times of up to 30 seconds whereas in Fanjul-Peyro and Ruiz (2010), DIG was shown to outperform CPLEX up to 60 seconds. The conclusion is that the new version of CPLEX makes a much more effective use of the faster computers.
The first two simple proposed size-reduction algorithms are fairly competitive. For longer CPU times, CPLEX is expectedly better as CPLEX is solving the full problem, but note how for short CPU times, i.e., 15 and 30 seconds, 3J2M outperforms CPLEX. Table 6 shows a more detailed result of the other three size-reduction algorithms: 3JD, 2JDi and 2JDi(90) together also with DIG and CPLEX.

The results of Table 6 show that, in average (right-most column), all of our proposed size-reduction algorithms produce better results than the state-of-art methods for all different stopping criteria. This table even has negative values because the algorithms, in these cases, reach better results than the reference values (CPLEX 11.0 with 2 hours stopping time). However, we can notice that, in the interval $U(1, 100)$, CPLEX produces good results for larger stopping times, improving 3JD and even 2JDi but still not reaching the values obtained by 2JDi(90). In general, when the stopping time exceeds the calibration value of 2JDi(90), which is 90 seconds, we notice how the 2JDi(90) algorithm reaches better results than its simpler counterpart algorithm 2JDi where the solver part is not stopped. The results show that CPLEX has a poor result in

correlated jobs but a good result in correlated machines, which it is the only case in which it can improve the results of our best algorithm for large CPU times. We can only say that this cannot be easily attributed due to the unknown structure of the specific algorithm that CPLEX is using for this assignment problem. In any case, CPLEX ends up being instance dependent and not robust as our proposed methods.

Time	Algorithms	$U(1,100)$	$U(10,100)$	Jobcorre	Machcorre	$U(100,200)$	$U(100,120)$	$U(1000,1100)$	**Average**
15	CPLEX	*0.82*	*0.70*	*1.33*	0.48	*0.55*	*0.09*	*0.05*	*0.58*
	DIG	1.07	0.62	0.45	*0.53*	0.29	0.04	0.02	0.43
	3JD	0.49	0.17	0.24	0.24	0.13	0.01	0.00	0.18
	2JDi	**0.44**	**0.17**	**0.24**	**0.22**	**0.13**	**0.01**	**0.00**	**0.17**
	2JDi(90)	**0.44**	**0.17**	**0.24**	**0.22**	**0.13**	**0.01**	**0.00**	**0.17**
30	CPLEX	0.56	0.47	*0.93*	0.30	*0.37*	*0.06*	*0.04*	*0.39*
	DIG	*0.93*	*0.49*	0.35	*0.50*	0.25	0.04	0.01	0.36
	3JD	0.37	0.08	0.13	0.23	0.11	0.01	0.00	0.13
	2JDi	**0.27**	**0.07**	**0.11**	**0.19**	**0.10**	**0.00**	**0.00**	**0.11**
	2JDi(90)	**0.27**	**0.07**	**0.11**	**0.19**	**0.10**	**0.00**	**0.00**	**0.11**
60	CPLEX	0.33	0.25	*0.65*	**0.16**	*0.25*	*0.04*	*0.03*	0.25
	DIG	*0.83*	*0.39*	0.25	*0.47*	0.21	0.03	0.01	*0.31*
	3JD	0.33	0.03	0.00	0.22	0.09	0.01	**0.00**	0.10
	2JDi	**0.19**	**0.01**	**-0.01**	0.17	**0.08**	**0.00**	**0.00**	**0.06**
	2JDi(90)	**0.19**	**0.01**	**-0.01**	0.17	**0.08**	**0.00**	**0.00**	**0.06**
120	CPLEX	0.17	0.10	*0.45*	**0.09**	0.13	*0.03*	*0.02*	0.14
	DIG	*0.75*	*0.24*	0.14	*0.42*	*0.17*	0.02	**0.00**	*0.25*
	3JD	0.31	0.01	-0.09	0.22	0.08	**0.00**	**0.00**	0.08
	2JDi	0.15	-0.06	**-0.11**	0.16	0.06	**0.00**	**0.00**	0.03
	2JDi(90)	**0.06**	**-0.07**	-0.10	0.15	**0.05**	**0.00**	**0.00**	**0.01**
240	CPLEX	0.08	0.05	*0.33*	**0.04**	0.06	*0.02*	*0.01*	0.09
	DIG	*0.63*	*0.17*	0.06	*0.37*	*0.14*	*0.02*	0.00	*0.20*
	3JD	0.26	-0.02	-0.13	0.22	0.07	0.00	0.00	0.06
	2JDi	0.10	-0.09	**-0.16**	0.15	0.06	0.00	0.00	0.01
	2JDi(90)	**-0.02**	**-0.14**	-0.15	0.11	**0.02**	**-0.01**	**-0.01**	**-0.03**
300	CPLEX	0.06	0.02	*0.29*	**0.03**	0.04	0.01	*0.01*	0.07
	DIG	*0.63*	*0.14*	0.03	*0.36*	*0.13*	*0.02*	0.00	*0.19*
	3JD	0.24	-0.07	-0.17	0.21	0.03	0.00	0.00	0.03
	2JDi	0.08	-0.11	**-0.19**	0.14	0.02	0.00	0.00	-0.01
	2JDi(90)	**-0.03**	**-0.14**	-0.17	0.10	**0.00**	**-0.01**	**-0.01**	**-0.04**

Table 6: Average Relative Percentage Deviations for serial algorithms CPLEX, DIG, 3JD, 2JDi and 2JDi(90) with different CPU time stopping criteria. Bold (italics) figures represent best (worst) results, respectively.

So far we have just shown average results. We need to carry out an ANOVA statistical test in order to guarantee that the observed differences in the average results are indeed statistically significant. Figure 1 represents a means plot of all intervals at 120 seconds stopping time. This

time is enough for CPLEX and for 2JDi(90) to reach good results. The means plot include Tukey HSD intervals with a 95% confidence level. Recall that overlapping intervals indicates that no statistically significant difference exists among the overlapped means. It can be seen CPLEX with this amount of time is significantly better than DIG. Our three proposed size-reduction algorithms are new state-of-the-art as can be seen, although there are some overlaps. 2JDi(90) is statistically better than 3JD but 2JDi is equivalent to the other two. However, this plot is for all intervals and instance sizes. Zoomed-in results (not shown due to reasons of space) show different performances depending on the cases.

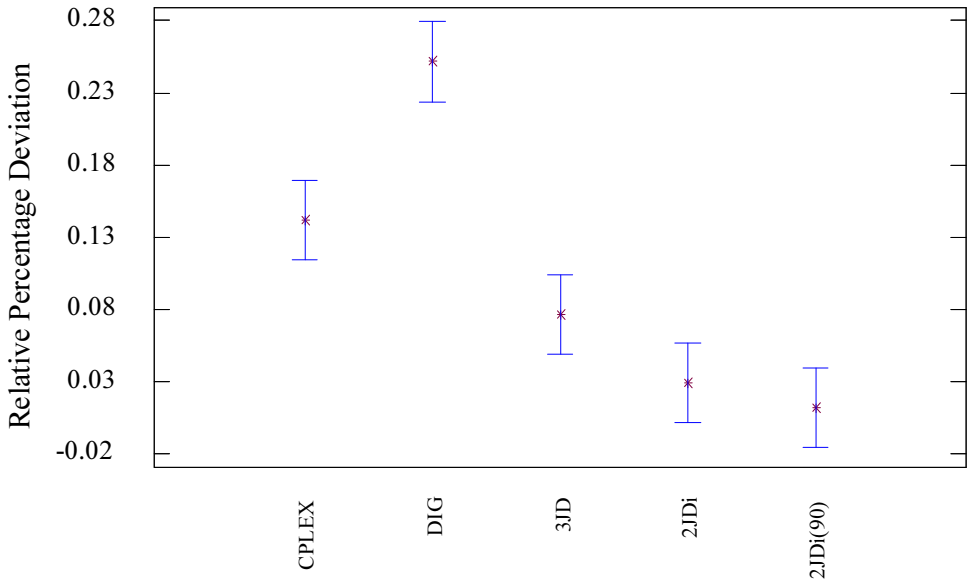

Figure 1: Means plot and Tukey HSD intervals with a 95% confidence level for the tested serial algorithms. All instances and intervals with 120 seconds stopping time.

4.2 Parallel algorithms' results

Now we test our proposed M-2JDi(90) parallel version of the 2JDi(90) size-reduction algorithm, along with parallel version of DIG and CPLEX (M-DIG and M-CPLEX, respectively). Results are given in Table 7. All parallel methods clearly outperform their serial counterparts by a significant margin if we compare these results against those given in Table 6. Recall that only two parallel cores are being used and probably better results are expected in four, six or even eight core settings. We see that, on average, our proposed parallel size-reduction algorithm produce better results than all other algorithms. M-DIG gives better results than M-CPLEX only for short CPU times and the situation is reversed when times are larger. Similar to the serial case, our algorithm M-2JDi(90) produces the best results in all cases and times except for correlated machines, where M-CPLEX has better results only for larger CPU times. M-CPLEX also presents poor results for correlated jobs as in the serial case.

Time	Algorithms	$U(1,100)$	$U(10,100)$	Jobcorre	Machcorre	$U(100,200)$	$U(100,120)$	$U(1000,1100)$	**Average**
15	M-CPLEX	0.66	*0.63*	*1.33*	0.37	*0.40*	*0.07*	*0.04*	*0.50*
	M-DIG	*0.91*	0.51	0.35	*0.48*	0.26	0.03	0.02	0.37
	M-2JDi(90)	**0.42**	**0.10**	**0.18**	**0.20**	**0.11**	**0.00**	**0.00**	**0.15**
30	M-CPLEX	0.41	0.33	*0.99*	0.20	*0.29*	*0.05*	*0.03*	*0.33*
	M-DIG	*0.86*	*0.41*	0.24	*0.45*	0.21	0.03	0.01	0.32
	M-2JDi(90)	**0.18**	**-0.01**	**0.03**	**0.16**	**0.08**	**0.00**	**0.00**	**0.06**
60	M-CPLEX	0.30	0.18	*0.74*	**0.09**	0.15	*0.03*	*0.02*	0.22
	M-DIG	*0.70*	*0.31*	0.16	*0.42*	*0.18*	0.02	0.01	*0.26*
	M-2JDi(90)	**0.10**	**-0.07**	**-0.07**	0.14	**0.06**	**0.00**	**0.00**	**0.02**
120	M-CPLEX	0.23	0.11	*0.60*	**0.04**	0.08	*0.03*	*0.01*	0.16
	M-DIG	*0.62*	*0.18*	0.07	*0.40*	*0.14*	0.02	0.00	*0.20*
	M-2JDi(90)	**-0.02**	**-0.14**	**-0.14**	0.12	**0.02**	**-0.01**	**-0.01**	**-0.03**
240	M-CPLEX	0.16	0.04	*0.47*	**0.02**	0.04	*0.01*	*0.00*	0.11
	M-DIG	*0.57*	*0.11*	-0.01	*0.34*	*0.11*	*0.01*	*0.00*	*0.16*
	M-2JDi(90)	**-0.03**	**-0.16**	**-0.19**	0.07	**0.00**	**-0.01**	**-0.01**	**-0.05**
300	M-CPLEX	0.08	-0.02	*0.35*	**0.01**	0.02	*0.01*	*0.00*	0.06
	M-DIG	*0.56*	*0.08*	-0.04	*0.34*	*0.10*	*0.01*	-0.00	*0.15*
	M-2JDi(90)	**-0.03**	**-0.17**	**-0.20**	0.05	**-0.01**	**-0.01**	**-0.01**	**-0.05**

Table 7: Average Relative Percentage Deviations for parallel algorithms M-CPLEX, M-DIG and M-2JDi(90) with different CPU time stopping criteria. Bold (italics) figures represent best (worst) results, respectively.

Figure 2 shows, in this case, an interaction plot between CPU time stopping criterion and the parallel algorithms. Note that a two factor ANOVA was used in this case. This ANOVA plot allows to see how M-DIG is significantly better than M-CPLEX for just 15 seconds stopping time. For 30 seconds both methods are statistically equivalent and they remain so until 300 seconds, where finally M-CPLEX improves the results of M-DIG. In all cases, our proposed size-reduction algorithm M-2JDi(90) is shown to statistically outperform the other two methods by a large margin.

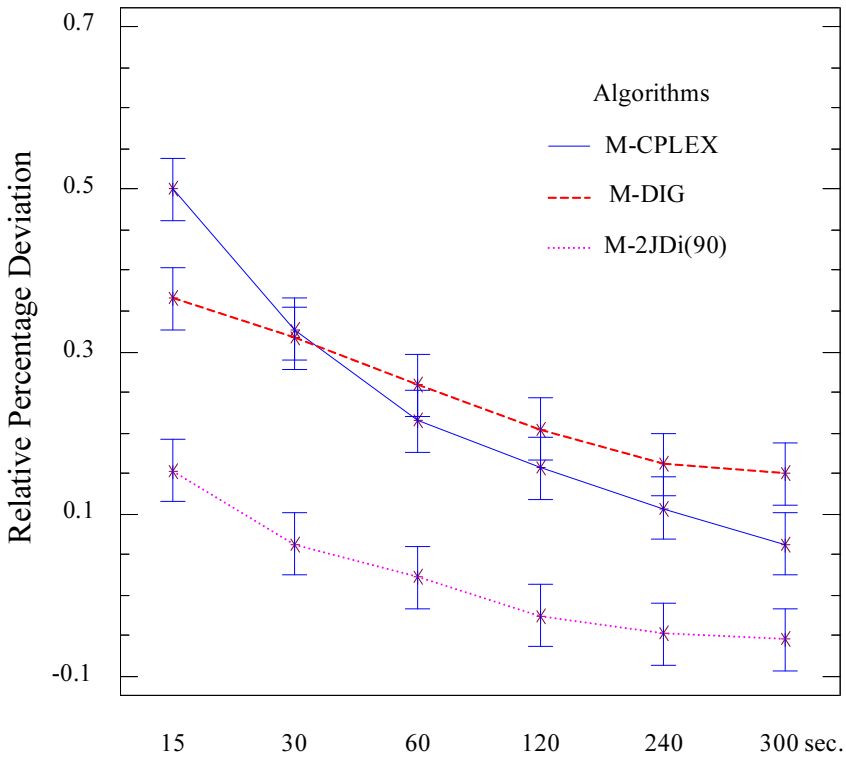

Figure 2: Means plot and Tukey HSD intervals with a 95% confidence level for the interaction between the stopping criteria and the tested parallel algorithms. All instances and intervals.

Finally, we present in Figure 3 a comparison between parallel and serial methods. Intervals are removed to improve readability of the plot. As we can see, parallel versions are most of the time better than serial versions. As expected, for large CPU times, most algorithms start to converge and the advantages of parallel methods diminish.

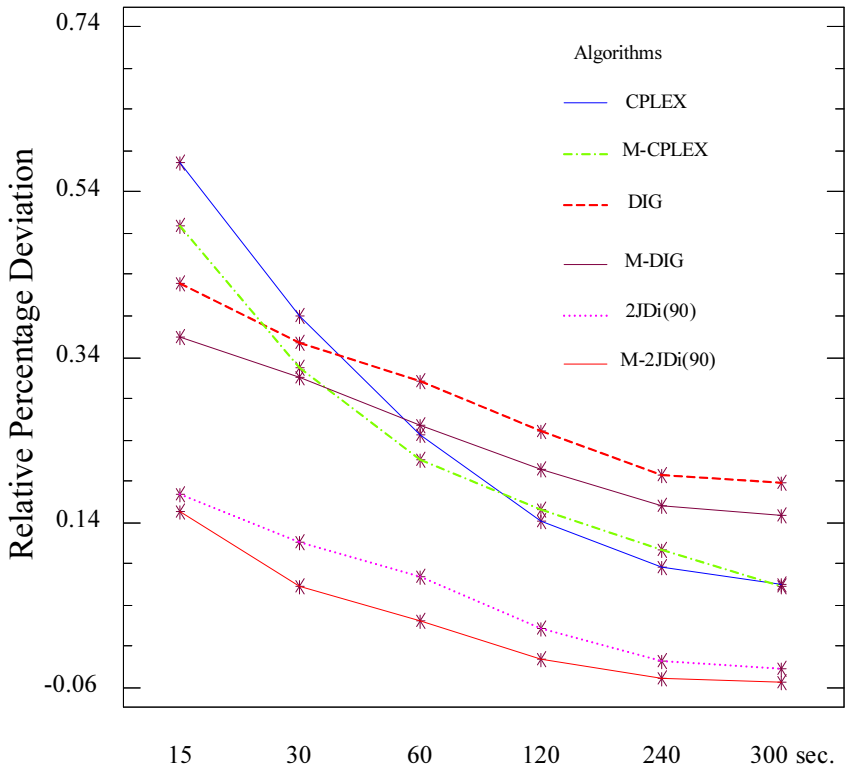

Figure 3: Parallel vs. serial algorithms: Means plot for all instances and intervals in all tested stopping times.

5 Conclusions and future research

In this paper we have proposed several new size-reduction methods and algorithms for the unrelated parallel machines scheduling problem with makespan criterion or $R//C_{\max}$. The methods presented are very simple and are based on the idea of reducing the possible assignment of jobs to machines to just the most promising ones. The reduced problem is then solved with the help of a commercial solver, which in this paper has been IBM-ILOG CPLEX version 11.1.

A comprehensive benchmark test of 1400 instances has been employed in order to compare all presented algorithms against state-of-the-art methods, identified as IBM-ILOG CPLEX 11.1, DIG from Fanjul-Peyro and Ruiz (2010) as well as parallel versions of these two methods. Our proposed methods outperform current state-of-the-art algorithms in comprehensive computational and statistical campaigns where both serial and parallel algorithms have been tested. More specifically, the parallel version of our best performing algorithm, M-2JDi(90), produces, in just 15 seconds of CPU time, average relative percentage deviations with respect to the best known reference lower bounds reported by the solver of only 0.63% across the 1400 instances that reach sizes of up to 1000 jobs and 50 machines. Note that this result is with respect the the lower

bounds, not against the best known reference solutions as reported in Table 7 which in this case are of 0.37%. This means that, in the worst case, 15 seconds suffice to reach an average deviation well below 1%. With these results we are very close to effectively solving the $R//C_{\max}$ problem to practical optimality for large sizes. Note however, that solving the $R//C_{\max}$ problem does not directly translate to solving the special cases with identical and uniform parallel machines, i.e., the $P//C_{\max}$ and $Q//C_{\max}$ cases, respectively. There are no guarantees that our proposed methods would perform equally well in these scenarios.

Future research stems from the consideration of more elaborated ways of obtaining the size-reduced problem although we hypothesize that given the already good results, such venue of research might result in diminishing returns. More interesting are the parallel approaches with more available cores. Real parallel methods with communications between the cores might help in solving much larger problems that might be of use for some applications. We are also interested in applying the proposed techniques to other more sophisticated parallel machines problems, like those resulting from the addition of sequence dependent setup times and/or to other objectives, like those based on job's due dates or just in time problems.

Acknowledgments

This work is partially funded by the Spanish Ministry of Science and Innovation, under the project "SMPA - Advanced Parallel Multiobjective Sequencing: Practical and Theoretical Advances" with references number DPI2008-03511/DPI. The authors should also thank the IMPIVA - Institute for the Small and Medium Valencian Enterprise, for the project OSC with reference IMIDIC/2008/137 and the Polytechnic University of Valencia, for the project PPAR with reference 3147.

References

Cheng, T. C. E. and Sin, C. C. S. (1990). A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47(3):271–292.

Davis, E. and Jaffe, J. M. (1981). Algorithms for scheduling tasks on unrelated processors. *Journal of the ACM*, 28(4):721–736.

De, P. and Morton, T. E. (1980). Scheduling to minimise makespan on unequal parallel processors. *Management Science*, 11(4):586–602.

de la Croce, F., Ghirardi, M., and Tadei, R. (2004). Recovering beam search: Enhancing the beam search approach for combinatorial optimization problems. *Journal of Heuristics*, 10(1):89–104.

Dillenberger, C., Escudero, L. F., Wollensak, A., and Wu, Z. (1994). On practical resource-allocation for production planning and scheduling with period overlapping setups. *European Journal of Operational Research*, 75(2):275–286.

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

Gairing, M., Monien, B., and Woclaw, A. (2007). A faster combinatorial approximation algorithm for scheduling unrelated parallel machines. *Theoretical Computer Science*, 380(1-2):87–99.

Garey, M. R. and Johnson, D. S. (1979). Computers and intractability: A guide to the theory of NP-completeness. *Freeman. San Francisco.*

Ghirardi, M. and Potts, C. N. (2005). Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach. *European Journal of Operational Research*, 165(2):457–467.

Glass, C. A., Potts, C. N., and Shade, P. (1994). Unrelated parallel machine scheduling using local search. *Mathematical and Computer Modelling*, 20(2):41–52.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326.

Hansen, P. and Mladenovic, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467.

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

Horowitz, E. and Sahni, S. (1976). Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM*, 23(2):317–327.

Ibarra, O. H. and Kim, C. E. (1977). Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM*, 24(2):280–289.

Lenstra, J. K., Rinnooy Kan, A. H. G., and Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362.

Lenstra, J. K., Shmoys, D. B., and Tardos, E. (1990). Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46(3):259–271.

Mansini, R. and Speranza, M. G. (1999). Heuristic algorithms for the portfolio selection problem with minimum transaction lots. *European Journal of Operational Research*, 114(2):219–233.

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

McNaughton, R. (1959). Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12.

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

Mokotoff, E. (2001). Parallel machine scheduling problems: A survey. *Asia-Pacific Journal of Operational Research*, 18(2):193–242.

Mokotoff, E. and Chretienne, P. (2002). A cutting plane algorithm for the unrelated parallel machine scheduling problem. *European Journal of Operational Research*, 141(3):515–525.

Mokotoff, E. and Jimeno, J. L. (2002). Heuristics based on partial enumeration for the unrelated parallel processor scheduling problem. *Annals of Operations Research*, 117(1-4):133–150.

Montgomery, D. C. (2009). *Design and Analysis of Experiments.* Wiley, New York, seventh edition.

Ow, P. S. and Morton, T. E. (1988). Filtered beam search in scheduling. *International Journal of Production Research*, 26(1):35–62.

Piersma, N. and 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.

Pinedo, M. L. (2005). *Planning and Scheduling in Manufacturing and Services.* Springer Series in Operations Research. Springer, New York, USA.

Pinedo, M. L. (2008). *Scheduling: Theory, Algorithms, and Systems*. Springer, New York, USA, third edition.

Potts, C. N. (1985). Analysis of a linear-programming heuristic for scheduling unrelated parallel machines. *Discrete Applied Mathematics*, 10(2):155–164.

Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049.

Shchepin, E. V. and Vakhania, N. (2005). An optimal rounding gives a better approximation for scheduling unrelated machines. *Operations Research Letters*, 33(2):127–133.

Shmoys, D. B. and Tardos, E. (1993). An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62(3):461–474.

Sourd, F. (2001). Scheduling tasks on unrelated machines: Large neighborhood improvement procedures. *Journal of Heuristics*, 7(6):519–531.

Srivastava, B. (1998). An effective heuristic for minimising makespan on unrelated parallel machines. *Journal of the Operational Research Society*, 49(8):886–894.

Sule, D. R. (2008). *Production Planning and Industrial Scheduling: Examples, Case Studies and Applications*. CRC Press, Boca Raton, USA, second edition.

van de Velde, S. L. (1993). Duality based algorithms for scheduling unrelated parallel machines. *ORSA Journal on Computing*, 5(5):192–205.

Woclaw, A. (2006). *Scheduling Unrelated Parallel Machines. Algorithms, Complexity, and Performance*. PhD thesis, Fakultat fur Elektrotechnik, Informatik und Mathematik der Universitat Paderborn, Deutschland.