4th International Conference on Industrial Engineering and Industrial Management
XIV Congreso de Ingeniería de Organización
Donostia- San Sebastián , September 8th -10th 2010

Size-reduction methods for the unrelated parallel machines problem and makespan criterion

Luis Fanjul-Peyro, Rubén Ruiz

Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática (ITI), Ciudad Politécnica de la Innovación, Edificio 8G, Acceso B, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022, Valencia, Spain. luifanpe@eio.upv.es, rruiz@eio.upv.es

Abstract

In this work we study the unrelated parallel machines problem with the objective of the minimization of the maximum completion time or makespan (C_{max}). We propose some metaheuristics based on a size-reduction of the original assignment problem. The idea is to consider only a few of the best possible machine assignments for the jobs and not all of them. We test the proposed simple 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 better by a significant margin.

Keywords: unrelated parallel machines, makespan, heuristics, size-reduction.

1. Introduction

In the parallel machines scheduling problem there is a set N of n jobs that have to be processed on a set M of m parallel disposed machines. Every job must be processed by exactly one out of the m parallel machines. In the most general case, the parallel machines are unrelated, i.e., the processing time of job j, $j \in N$ depends on the machine i, $i \in M$ to which it is assigned to. As a result, the input data to the problem are the number of jobs n, the number of machines m and a processing time matrix p_{ij} . The problem consists on the assignment of jobs to machines and then on the scheduling of all the assigned jobs at each machine. One of the most commonly studied optimization criteria is the minimization of the maximum completion time or C_{max}. Using the well known α/β/γ classification of Graham et. al. (1979), this problem is denoted as $R//C_{max}$. This is a *NP*-Hard problem in the strong sense, since Garey and Johnson (1979) demonstrated that the simpler identical parallel machines case is already *NP*-Hard in the strong sense. Lenstra et. al. (1977) showed that the special case of two identical machines is also *NP*-Hard. Notice that with the C_{max} objective, the $R//C_{max}$ is just a type of assignment problem since the sequence of the jobs assigned to any machine does not influence the value of the C_{max}. To fully define this problem we have a well known Mixed Integer Linear Programming (MILP) assignment model, where the variables x_{ij} are one if job j is assigned to machine i and zero otherwise. The model goes as follows:

$$\min \quad C_{max} \quad s.t. \quad \sum_{i=1}^{m} x_{ij} = 1 \quad \forall j \in N; \quad \sum_{j=1}^{n} x_{ij} \cdot p_{ij} \leq C_{max} \quad \forall i \in M$$

Many papers about the $R//C_{max}$ problem have been published since the original work of McNaughton (1959). We can find several reviews like the ones of Cheng and Sin (1990) and Mokotoff (2001). A more recent and updated review is given by Fanjul-Peyro and Ruiz (2010) (available at the publications section of http://soa.iti.es) where it was shown that state-

of-the-art results can be obtained by a simple iterated greedy metaheuristic method referred to as NVST-IG+ (denoted here as DIG for simplicity). The same authors also demonstrated that new versions of commercial solvers, and more specifically, IBM-ILOG CPLEX version 11.1 (denoted here as CPLEX) are extremely competitive in finding optimum solutions just using the previous MILP model. In this short paper we propose size-reduction methods that take advantage of modern solvers in order to achieve even better results.

2. Size-reduction algorithms

We observed that CPLEX is very capable of solving some instances of considerable size (up to 1000 jobs and 50 machines). However, CPLEX needed large CPU times to solve big instances and large CPU times to prove the optimality of the solutions. A close analysis of the resulting sequences yielded an expected but rather surprising result: In almost all situations, a given 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 the machines i_1, i_2 or i_3 where i_1=argmin $_{j \in M} p_{ij}$; i_2=argmin $_{j \in M \setminus i_1} p_{ij}$ and i_3=argmin $_{j \in M \setminus i_1, i_2} p_{ij}$. The resulting "reduced" MILP model has $n \cdot 3$ binary variables. This means that if a 1000×50 instance needs 50,000 binary variables, by just using the "three fastest machines" for each job all we need is 3,000 variables, i.e., 16.67 times less variables. In order to test this proposal, we count how many jobs were not assigned to any of the three fastest machines i_1, i_2 or i_3 in the instances used by Fanjul-Peyro and Ruiz (2010) which employed a total of 1,400 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 and for the reference solutions obtained with CPLEX with a 2 hours termination criterion on each instance. Further details about this benchmark will be given later. Results are given in Table 1. Each cell in Table 1 represents the percentage of jobs in the instance that were assigned to a machine not in i_1, i_2 or i_3 in the final mentioned reference solution.

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%. 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.

Of course, this size reduction comes at a high cost. First of all, optimality cannot be guaranteed and second, this straightforward size reduction does not perform that well in all cases. A less naïve approach is needed. We devised several size reduction methods. The first has been already explained and consists in selecting i_1, i_2 and i_3 for every job. We call this the 3J size-reduction method. We also use a selection within a machine calculating the theoretical average of jobs that should be assigned to each machine, i.e., $a = n/m$, and selecting two times ($2 \cdot a$) the smallest values of p_{ij} at each machine. This last method, used together with 3J, is denoted by 3J2M.

Given the good results obtained with the DIG algorithm in Fanjul-Peyro and Ruiz (2010), a direct idea is to use the p_{ij} values obtained after analyzing all the best solutions obtained by DIG in a number of iterations. This procedure is used together with 3J to form what we call the 3JD size-reduction method.

Table 1. Percentage of jobs at each instance that are not assigned to any of the three fastest machines i_1, i_2 or i_3 in the reference solutions given by IBM-ILOG CPLEX 11.0 with a time limit of 2 hours.

n	m	U(1,100)	U(10,100)	U(100,200)	U(100,120)	U(1000,1100)	Jobcorre	Machcorre
100	10	0.3	0.5	0.8	1.3	1.1	0.7	30.2
	20	1.4	1.2	4.6	1.5	2.8	4.4	35.4
	30	3.8	4.6	18	24.4	31.2	10.9	41.3
	40	3.5	7.3	20.4	11.1	24.7	17.9	43.8
	50	5.5	10.2	6.6	1.5	4.8	32.4	41.3
200	10	0.2	0.1	0.3	0.3	0.5	0.3	31.9
	20	0.4	0.65	1.4	0.55	1.3	2.2	40.3
	30	0.8	1.65	6.8	17.75	10.5	4.75	44.05
	40	1.2	1.85	1.85	0.6	3.2	9.9	45.75
	50	2.15	3.65	4.3	0.25	4.35	12.6	46.15
500	10	0.04	0	0.06	0.1	0.08	0.02	32.82
	20	0.08	0.12	0.3	0.2	0.24	0.18	43.14
	30	0.32	0.12	2.34	13.46	8.8	0.8	44.48
	40	0.52	0.28	4.46	11	10.12	2.1	44.3
	50	0.36	0.74	0.84	0.28	0.84	4.2	45.1
1000	10	0.02	0.01	0	0.02	0.02	0	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	0.2	0.11	0.45	0.1	0.22	1.33	45.14
Average(%)		1.05	1.66	3.74	4.32	5.88	5.29	40.45

Once the original problem has been reduced by the usage of 3J, 3J2M, or 3JD, we simply run CPLEX until the optimum solution is found for the reduced problem. Obviously, it is not guaranteed that this optimum solution is also optimum for the original problem. Some further refinements can be implemented. If the solver reaches an optimal solution of a size-reduced problem before the allotted CPU time has elapsed, we can increase the number of p_{ij} to be considered (more binary x_{ij} variables) and re-launch the solver with this increased size-reduced problem. An algorithm results from this procedure: start by selecting only the two smallest p_{ij} values for each job (i_1 and i_2) and each time an optimal solution for the size-reduced model is reached, we select another non previously selected smallest p_{ij} for each job and re-launch the solver. This is repeated until the termination criterion is met. We denote this algorithm as 2JDi.

Another refinement comes from the fact that the solver, in most cases, uses a lot of time to close the search tree when proving optimality, without improving the solution. For this reason, instead of solving each size-reduced model to optimality, we solve each model with a maximum CPU time limit. When this limit is reached, we re-launch an increased size-reduced model with the same time limit starting from the previously best known solutions. After some calibrations, we fix this CPU time limit at 90 seconds and denote this algorithm as 2JDi(90).

Current CPLEX versions run in parallel, using the available multiple cores of modern computers. We also develop some simple parallel versions of our algorithms. For example, the previous 2JDi(90) method is "parallelized" by just running the first size-reduced and smallest problem in one core and the next increased sized-reduced model in the second core. Whenever a core finishes, the next pending increased size-reduced model is launched. We denote this algorithm as M-2JDi(90) (Multicore 2JDi(90)). We also generate a simple parallel version of the DIG algorithm by just running as many threads of the DIG, each one running in one core, and after the termination criterion is reached, the best solution found in any core is

retained. Lastly, CPLEX is checked both in single-core and multi-core, denoted as CPLEX and M-CPLEX respectively.

3. Computational analysis

We use a comprehensive set of benchmark instances for the $R//C_{\max}$ proposed by Fanjul-Peyro and Ruiz (2010) that are composed by 7 different intervals for the distribution of the p_{ij} processing times: U(1,100), U(10,100), U(100,200), U(100,120), U(1000,1100), correlated jobs where p_{ij} are determined by $p_{ij} = b_j + d_{ij}$ where b_j and d_{ij} come from U(1,100) and U(1,20), respectively and correlated machines with $p_{ij} = a_i + c_{ij}$ where a_i and c_{ij} are uniformly distributed as U(1,100) and U(1,20), respectively. For each interval, we have the following combinations of n and m: n={100,200,500,1000}, m={10,20,30,40,50}. For each combination there are 10 replicates. There are 200 instances at each interval and 1,400 instances in total.

All tests have been run 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 and under Windows XP SP3 operating system. Notice that these computers have two cores. In order to compare the results of the algorithms we use as reference values those given by Fanjul-Peyro and Ruiz (2010), which were obtained after solving each instance with the solver IBM-ILOG CPLEX 11.0 with a 2 hour maximum CPU time limit. The response variable is the relative percentage deviation from this reference solution as follows:

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

Where $C^*_{\max}(i)$ is the aforementioned 2 hour CPLEX solution (many times optimal or with a very small gap) and $C_{\max}(i)$ is the value obtained by a given algorithm and instance i. We also carry out ANOVA tests in order to guarantee that the observed differences in the average results are indeed statistically significant. The computational analysis is divided between the serial and parallel methods.

3.1. Serial algorithms' results

Our first test comprises the state-of-the-art algorithm DIG proposed by Fanjul-Peyro and Ruiz (2010) and the CPLEX version 11.1. We put along the first simple algorithms 3J and 3J2M. Table 2 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 1,400 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.

Table 2. Average Relative Percentage Deviations for all intervals except correlated machines for existing state-of-the-art and simple size-reduction algorithms 3J and 3J2M.

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

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. Now we test in more detail the more advanced processed methods, whose results are given in Table 3.

The results of Table 3 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.

Table 3. 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.

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**

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.

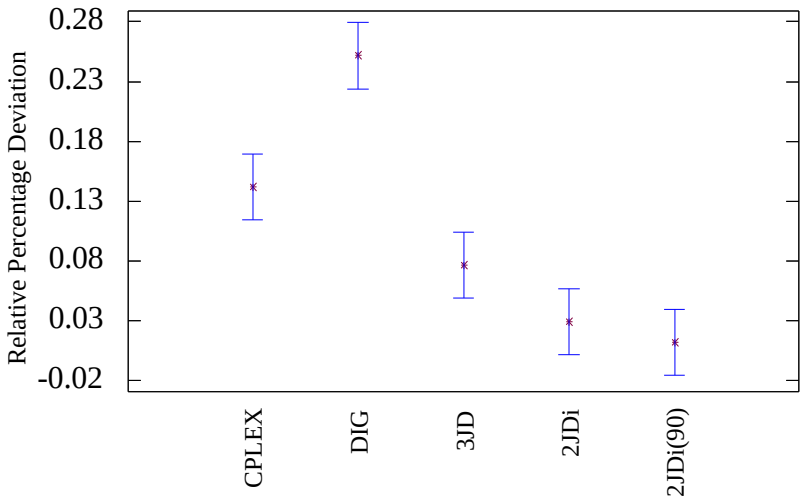

Figure 1. Means plot and Tukey HSD intervals with a 95% confidence level for the tested serial algorithms. All instances and intervals with 120 seconds stopping time.

3.2. 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 4. All parallel methods clearly outperform their serial counterparts by a significant margin if we compare these results against those given in Table 3. 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.

Table 4. 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.

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**

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.

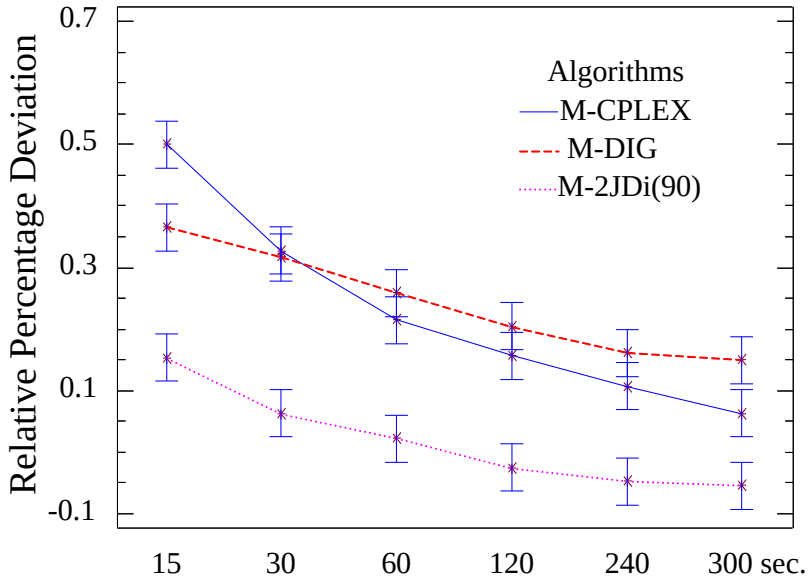

Figure 2. Parallel methods: Means plot and Tukey HSD intervals with 95% confidence level for all instances and intervals and all tested stopping criteria.

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.

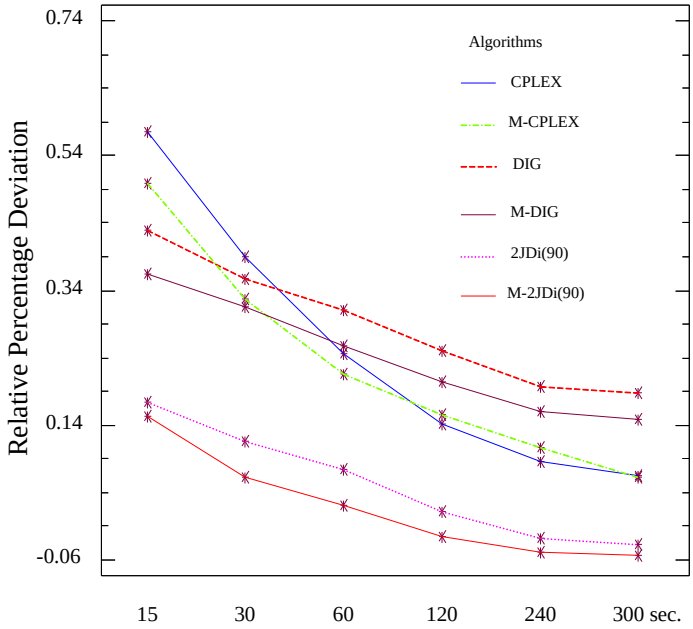

Figure 3. Parallel vs. serial algorithms: Means plot for all instances and intervals in all tested stopping times.

We can conclude that our algorithms show a very good performance in all situations, serial, parallel, with different stopping times and under a large and comprehensive benchmark. These results are remarkable, especially if we consider the inherent simplicity of the proposed methods.

4. Conclusions

In this paper we have shown simple size-reduction methods that when coupled with modern solvers result in state-of-the-art performance for the makespan minimization unrelated parallel machines scheduling problem in both scenarios, serial and parallel. The methods presented are very simple as they are just based on the idea of reducing the number of binary variables in the so common Mixed Integer Linear Programming (MILP) assignment model.

Finally, we want comment that 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 1,400 instances that reach sizes of up to 1000 jobs and 50 machines. With these results we are very close to effectively solving the $R//C_{\max}$ problem to practical optimality for large sizes.

Acknowledgements

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 reference 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, Vol. 47, No. 3, pp. 271-292.

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.

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

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, Vol. 5, pp. 287-326.

Lenstra, J. K., Rinnooy Kan, A. H. G., and Brucker, P., 1977, "Complexity of machine scheduling problems", Annals of Discrete Mathematics, Vol. 1, pp. 343-362.

McNaughton, R.,1959, "Scheduling with deadlines and loss functions", Management Science, Vol. 6, No. 1, pp. 1-12.

Mokotoff, E.,2001, "Parallel machine scheduling problems: A survey", Asia-Pacific Journal of Operational Research, Vol. 18, No. 2, pp. 193-242.