Fanjul Peyró, L.; Ruiz García, R. (2012). Scheduling unrelated parallel machines with optional machines and jobs selection. Computers and Operations Research. 39(7):1745-1753. doi:10.1016/j.cor.2011.10.012.

Scheduling unrelated parallel machines with optional machines and jobs selection

Luis Fanjul-Peyro, Rubén Ruiz*

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

Ciudad Politécnica de la Innovación, Edificio 8G. Acceso B. Universitat Politècnica de València,

Camino de Vera s/n, 46022 Valencia, Spain, {lfpeyro,rruiz}@eio.upv.es

October 11, 2011

Abstract

In this paper we study two generalizations of the well known unrelated parallel machines scheduling problem under makespan ($C_{\max}$) minimization. First, a situation in which not every available parallel machine should be used and it is desirable to employ only a subset of the parallel machines. This is referred to as "Not all Machines" or NAM in short. This environment applies frequently in production shops where capacity exceeds demand or when production capacity can be lent to third companies. Also, NAM can be used to increase production capacity and it is not clear how many additional machines should be acquired. The second studied generalization has been referred to as "Not All Jobs" or NAJ. Here, there is no obligation to process all available jobs. We propose Mixed Integer Programming mathematical formulations for both NAM and NAJ, and it is shown that the latter can be effectively solved with modern commercial solvers. We also present three algorithms to solve the NAM problem. These algorithms are compared with the proposed MIP formulation when solved with IBM ILOG CPLEX 12.1. Comprehensive computational and statistical experiments prove that our proposed algorithms significantly improve the results given by the solver.

Keywords: unrelated parallel machines, makespan, optional machines, not all machines, job selection, not all jobs

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

1 Introduction

There is a set M of m machines that are arranged in parallel. Each job from a set N of n jobs has to be processed by exactly one machine. Each machine cannot process more than one job at the same time. Furthermore, preemption of jobs is not allowed and when a job begins its processing, it cannot be stopped until its completion. The previous problem is known as parallel machine scheduling and is divided into three different cases, which depend on the nature of the available parallel machines. The simplest scenario is when all parallel machines are identical. In this case, each job j, $j \in N$ needs a fixed, non-negative, and known in advance processing time, denoted as p_j. Once started in the assigned machine, this machine will be busy processing job j during p_j time units. The particularity in the identical parallel machines scheduling case is that the processing time for each job j is the same for all machines. A more general case is the uniform parallel machines scheduling problem. Here, the processing time of a job j depends on the machine i, $i \in M$ to which it is assigned. However, this dependence follows a strict speed-up factor s_i that varies from machine to machine. Therefore, processing times follow the relationship $p_{ij} = p_j/s_i$. Higher s_i values indicate "faster" machines and this speed-up factor is fixed for each machine. The most general situation is referred to as unrelated parallel machines scheduling. In this case, each machine processes each job at a different speed and p_{ij} denotes the processing time of each job j which depends on the machine i to which it is assigned.

This paper deals with this last and most general production scheduling scenario. The optimization criterion considered is the minimization of the maximum completion time, commonly referred to as makespan and denoted by $C_{\max}$. This scheduling problem is denoted by $R//C_{\max}$, following the well known three-field notation $\alpha/\beta/\gamma$ of **?**. While the $R//C_{\max}$ is classified as a scheduling problem, the job sequence followed at each machine has no influence over the final $C_{\max}$ value. The whole setting reduces to an assignment problem. This is easily explained as follows: J_i denotes the subset of jobs ($J_i \subseteq N, \forall i \in M$) that have been assigned to machine i. Therefore, if we assume that machine i is available from time 0, it will be busy during $C_i = \sum_{\forall k \in J_i} p_{ik}$ units of time. Given all $J_i, \forall i \in M$, the makespan is easily defined as the time at which the last machine is free or $C_{\max} = \max_{i \in M}\{C_i\}$. It is straightforward to see that the order in which jobs assigned to machine i (J_i) are processed has no influence over C_i and therefore, the $C_{\max}$ value depends solely on the job to machine assignments. Note that this result applies only to the makespan optimization criterion. Given all possible assignments, the cardinality of the feasible solution set is no less than m^n. In fact, the $R//C_{\max}$ problem is an $\mathcal{NP}$-Hard problem in the strong sense,

after ? demonstrated the special case with identical machines ($P//C_{\max}$) to be $\mathcal{NP}$-Hard. Additionally, even before, ? demonstrated that the two machine version, or $P2//C_{\max}$, was already $\mathcal{NP}$-Hard.
The $R//C_{\max}$ problem has many potential applications. Mass production lines usually contain more than one machine for each production task. As mass production lines are ubiquitous, parallel machines scheduling settings are equally frequent in practice. Other examples are multiprocessor computers, landing lanes at airports or even operating rooms in hospitals, which can also be seen as parallel machine shops. Many other examples and applications can be obtained from general scheduling textbooks such as ?, ?, or ?.

The $R//C_{\max}$ problem has been comprehensively studied since the first paper of ?. Some general parallel machines scheduling review papers have been published, like those of ? and ?. ? recently presented an updated short review on state-of-the-art methods. Also recently, ? proposed simple but highly effective methods providing average deviations of just 0.63% with respect to tight lower bounds in as little as 15 seconds of CPU time. These results were obtained for large benchmark sets of 1400 instances, spread over 7 groups with different intervals of processing times and with sizes up to 1000 jobs and 50 machines. After such small deviations from lower bounds in such small CPU times, it is safe to state that, as regards practical applications, the $R//C_{\max}$ problem is quite well solved nowadays. This is not to say that it is a solved problem, of course, since it belongs to the $\mathcal{NP}$-Hard complexity class. For even larger problems, good solutions might still prove challenging to obtain.

Given the previous recent developments, it is quite natural to extend the $R//C_{\max}$ problem into new directions. In this paper we deal with two generalizations of the unrelated parallel machines problem. In the first we have that not all machines in the set M can be used and some machines have to be left out. More specifically, among the m available machines, a number Z of them cannot be used. This seemingly straightforward extension, has not been, to the best of our knowledge, and as we will later show, studied up to date as regards the $R//C_{\max}$ specific problem. We have referred to this generalization as the "Not All Machines" or NAM problem. It is very common to find workshops where an excessive production capacity exists and a tactical decision arises as to which machines should be stopped from set M. Unrelated parallel machines process each job at a different speed and deciding which machine or machines have to be stopped goes way beyond simply stopping the slowest ones. Other potential applications of the NAM problem appear if one considers the symmetric problem, i.e., there is a lack of production

capacity and more machines need to be purchased and/or subcontracted. A large set of potentially new machines can be added with the constraint of just using a given number of additional machines. Again the question is which new machines are to be used. Problems similar to NAM appear in the literature, for example where machines are subject to known unavailability periods. However, the problem in which, let us say, a workshop with 10 parallel machines are available and when it has been decided that 4 machines should be stopped has not been approached in the literature. The closest references that we have been able to identify in the literature are the papers of **?**, **?** or **?**. However, these studies deal with identical machines or jobs with unitary/identical processing times and with different objectives or restrictions. Other studies, like the one of **?** consider a cost function, together with job tardiness in a weighted objective function, and simultaneously select machines and optimize tardiness values. More recently, **?** studied regular performance measures and machine cost (and selection) considerations but for the more specific case of identical parallel machines. The reader is referred to this more recent paper for additional references.

The second generalization studied in this paper is when not all jobs in the set N need to be processed. We have dubbed this extension as "Not All Jobs" or NAJ. In details, we have a total of n jobs and only H, $H < n$, jobs have to be processed, discarding the remaining $n - H$ jobs in the process. NAJ-like settings have been thoroughly studied at a more production planning stage under various names like order acceptance, due date setting or even Just In Time (JIT) scheduling. Good reviews of due date setting and JIT scheduling are given by **?** and **?**, respectively. However, and again to the best of the knowledge of the authors, it has not been studied together with the parallel machines production scheduling problem. The NAJ problem appears quite frequently at companies where there is the possibility of not accepting –or not producing in the current production planning horizon– a given subset of jobs. The possible benefits and applications of NAJ are manifold. Selecting only profitable products or products that employ a given under-utilized machinery are common examples. Note that NAJ is very similar to other studied settings just as the already mentioned JIT. Nevertheless, there are basic differences between these scenarios. For example, in JIT problems, jobs are usually assigned a deadline and after the scheduling algorithm is applied, jobs not able to finish by their deadline are usually discarded. In the NAJ setting there are no deadlines and the idea is to select a subset of jobs not worth doing or likewise, to select a set of jobs worth producing.

? and **?** showed that powerful commercial solvers such as IBM ILOG CPLEX versions

11.0 and 11.1 obtain excellent solutions for the unrelated parallel machines scheduling problem, much better than the methods that were considered state-of-art at the time. As a result, it is reasonable to start first with mathematical programming models for NAM and NAJ, which are developed in Section 2. After the formulation we present three algorithms to solve the NAM setting along with computational and statistical analyses of performance in Section 3. The performance of existing solvers, in this paper IBM ILOG CPLEX 12.1, is also checked. In section 4 we study the NAJ setting. Finally, conclusions and further research opportunities are presented in Section 5.

2 Mathematical programming formulation

We begin with a straightforward Mixed Integer Linear Programming (MILP) formulation for the $R//C_{\max}$ which is as follows:

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

$$\sum_{j=1}^{n} p_{ij} \cdot x_{ij} \leq C_{\max}, \quad \forall i \in M \tag{2}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \quad \forall j \in N \tag{3}$$

where x_{ij} is a binary variable which takes value 1 if job j is assigned to machine i and 0 otherwise. The set of restrictions (2) assign the $C_{\max}$ value which can not be lower than C_i for each machine. Constraints in the set (3) ensure that all jobs are assigned to exactly one machine.

The modifications for the NAM setting are simply to add the following sets of constraints:

$$\sum_{j=1}^{n} x_{ij} \leq n \cdot z_i, \quad \forall i \in M \tag{4}$$

$$\sum_{i=1}^{m} z_i \leq m - Z \tag{5}$$

where z_i is a new binary variable which takes value 1 if machine i is used and 0 otherwise. Note that in constraint set (4) the maximum number of assigned jobs to any

machine is n but only if the machine is used ($z_i = 1$). The single constraint (5) limits the maximum number of machines to be used to $m - Z$, where Z denotes the machines to be left out unused, as already stated. In total we need m additional binary variables and $m + 1$ additional constraints for modeling the NAM problem.

The NAJ modifications involve the following sets of constraints:

$$\sum_{i=1}^{m} x_{ij} = h_j, \quad \forall j \in N \tag{6}$$

$$\sum_{j=1}^{n} h_j \geq H \tag{7}$$

where h_j is a new binary variable which takes value 1 if job j is processed and 0 otherwise. The set of restrictions (6) replace the previous set (3). Again we have a single constraint (7) that sets the minimum number of jobs to be processed to H.

Note that the NAM and NAJ generalizations are not necessarily tied to makespan minimization.

3 Methods for solving the "Not All Machines" (NAM) generalization

Two parts can be distinguished in the NAM problem. The first one is to decide which machines will not be used and the second is to solve the resulting parallel machines scheduling problem without these machines. In the case of unrelated parallel machines, the decision of which machines should be left out is not an easy task since the processing time of each job depends on the machine and there are no machines that are consistently slower or faster for all jobs (this would be the less general case of uniform parallel machines). As a result of this, if one aims to optimally solve this problem, it would be necessary to find the best possible combination of machines to be employed. This is a combinatorial problem where all possible combinations of $m - Z$ machines that are selected for use (not using Z machines) must be chosen among the total m machines. For example, in a problem with 10 machines, if we only want to use 7 of these machines, we would have a number of combinations without repetition with 10 machines taken 7 at a time to select the machines that are used or, equivalently, 10 elements taken 3 at a time to select the machines not to be used. The result is $\binom{10}{7} = \binom{10}{3} = 120$ possible combinations. Each combination results

in a different $R//C_{\max}$ problem that should be solved to optimality. Obviously, this is a problem of significant dimensions, since for a more realistic example with a workshop of 50 parallel machines where we wish not to use 10 of them we would have a total of 10,272,278,170 possible combinations. Given the impossibility of solving this problem optimally, we propose a heuristic approach that arises from decomposing the NAM problem into three phases:

1. Analyze the processing times and rank the most promising machines.
2. Selection of machines according to the ranking.
3. Solve the resulting $R//C_{\max}$ problem.

The last two phases can be iteratively applied until a given stopping criterion is reached, since different machine selections will result in different $R//C_{\max}$ problems.
Once we have solved the resulting problem, we must take into account that perhaps the selected machines were not the best ones. Therefore, we must follow a certain criterion for not only making a first selection of machines, but also to make successive selections in the hope that better solutions may be found. For example, if we have a problem with 100 jobs and 10 machines on which only eight of them must be used, maybe our best first option is not to use machines 1 and 2. After solving the resulting unrelated parallel machines problem with 100 jobs and 8 machines, we can select two different machines, for example 1 and 3 in the ranking. With this new set of machines we solve the problem again. Each time we minimize the makespan of the resulting problem.
The three phases of the NAM generalization solution procedure are explained in the following sections in more detail.

3.1 Machine ranking procedure

As a first step, we devise a procedure to rank machines to identify which ones are potentially interesting. The obvious first ranking choice is to solve m problems, each one of them with $m-1$ machines after removing each potential machine. One less machine forces the other machines to have higher work loads and therefore the makespan will increase. The worst makespan obtained among the m problems gives an indication of a machine that should not have been removed. Conversely, the lowest makespan obtained indicates a machine that is not that much needed. After this removal, we have only $m-1$ remaining machines. Repeating the process by removing another machine and solving $m-1$ problems with $m-2$ machines each results in another machine candidate. After repeating

this process Z times –which means solving no less than $Z \cdot m - \frac{Z \cdot (Z-1)}{2}$ unrelated parallel machines problems– we might have a good ranking of machines to be removed. However, this process, apart from being extremely slow, gives very bad results since machines are being removed in a greedy way one at a time instead of considering them all together. Therefore, a more efficient and effective procedure for ranking machines is recommended.

? observed that good solutions for the $R//C_{\max}$ problem contain job-machine assignments where most of the time jobs are assigned to the first, second or third machine with the lowest processing time. Following the same idea we propose a simple ranking. We extract the three lowest values of processing times p_{ij} for each job j, i.e., i_{1j}, i_{2j} and i_{3j}. To each one of these three values we subtract the fourth lowest one ($i_{4j} = \min_{i \in M/i_{1j},i_{2j},i_{3j}} p_{ij}$). The result is a negative value. In general, negative results give us an indication of how much faster it is to process a job in the first three machines when compared to the fourth. Note that the fourth machine is just a reference machine and similar rankings could be obtained using the fifth or subsequent machines. This is calculated for each job. Finally, for each machine, we add the values resulting from each subtraction. Machines are sorted in ascending order of this amount, where ties are broken arbitrarily. The first machine in the ranking is, in general, the machine which has, on average, shorter processing times for all jobs. Note that this procedure is an implicit weighting scheme. The higher the differences between the considered processing times, the higher the difference and the higher the machine will be ranked. Notice that a machine may not have any of the first three minimum processing times for any job. In these cases, instead of adding the previous subtractions, we add all the original processing times of all jobs.

Let us illustrate the machine ranking procedure by means of an example. Table 1 contains the processing times of a 10 job, 5 machine $R//C_{\max}$ example. For each job, the three lowest processing times have been highlighted in italics whereas the fourth lowest processing time is marked in bold.

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

Table 1: $R//C_{\max}$ example problem. Processing times p_{ij} for a problem with 10 jobs (columns) and 5 machines (rows). The three lowest processing times for each job in italics and the fourth in bold.

Now we proceed to subtract the fourth lowest processing time from each one of the three lowest processing times. This is shown in Table 2. The C_i column contains sum of the values thus obtained for each machine. C_o equals C_i in the case that C_i is not a zero. If C_i is zero, then C_o is equal to the sum of all original p_{ij} values for machine i from Table 1. Following the example, the ranking would be $\{M5, M4, M2, M3, M1\}$. This means that machine M5 is the "most needed" machine as regards makespan minimization whereas machine M1 is the most expendable one.

	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10	C_i	C_o
M1											0	47
M2	-2	-1			-2	-2		-2	-1	-3	-13	-13
M3		-1	-3	-4	-1	-1	-1				-11	-11
M4	-2	-1	-3	-1			-3	-1	-3	-2	-16	-16
M5	-2		-3	-2	-1	-1	-1	-2	-3	-2	-17	-17

Table 2: Difference between the three lowest processing times for each job and the fourth lowest one, total sums and corrected sums.

3.2 Machine selection

Once the ranking for the machines has been calculated, we proceed to make a selection of the machines to use. After each selection, we solve the resulting problem with the selected machines.

There are m machines and we want to select only $m - Z$, where Z is the number of machines that will not be used in the workshop. The previous ranking is already calculated. We propose the following selections:

1. The first selection contains the $m - Z$ first machines of the ranking.

2. In the second selection, we choose the machine in position $m - Z + 1$ from the ranking, i.e., the first unselected machine, and exchange it with the last selected machine, which is located in position $m - Z$ from the ranking.

3. For the third selection, with respect to the first selection, we exchange the first unselected machine again, in position $m - Z + 1$, with the machine in position $m - Z - 1$ of the ranking.

4. The process continues until the first unselected machine has been exchanged with all selected machines of the first selection.

5. The above process is repeated again but this time exchanging the second unselected machine, i.e., the machine that occupies the position $m - Z + 2$ of the ranking, with all machines selected in the first selection. Afterwards we proceed with the third unselected machine and so on until all unselected machines have been tested.

The previous list is very detailed in order to have a clear description of the selections. However, the selection is carried out in two nested loops and, in fact, these steps are easier described as in a cycle k, where $k = 1, 2, \ldots, Z$, machine $m - Z + k$ in the ranking is selected and swapped in turn with each one of the first $m - Z$ machines.

In total, there are $(m - Z) \cdot Z + 1$ selections or different sets of machines. All these sets are potentially good sets as a result from the machine ranking procedure. Note that each selection generates a different $R//C_{\max}$ problem that needs to be solved. As already commented, and as we will later show, the proposed algorithms first carry out a selection according to the previous list and then solve the problem. The next selection is carried out and the problem is solved again. If all previous selections are calculated and there is still time left for carrying over, successive selections are just made of random machines uniformly selected. Basically, we need this random machine selection phase for the only reason of not having the algorithm stop before a predefined CPU time, but our results (to be discussed later) indicate that there are little to no gains with this last random machine selection. Additionally, the limited allotted CPU time results in random machine

selections only for instances with a small number of machines.

Let us follow the previous example. Recall that the machine ranking was $\{M5, M4, M2, M3, M1\}$. We have that 40% of the machines in the shop have to be stopped ($Z = 2$), which means that we have to select the $m - Z = 5 - 2 = 3$ best machines. The first ranking is therefore $\{M5, M4, M2\}$. Notice that these machines are the three most promising ones according to the ranking. Once this three machine problem is solved and a makespan value obtained, a second selection is carried out. We take the first unselected machine according to the ranking, machine $M3$, and exchange it with the last selected machine, $M2$. This means that the machines to use in the second selection are $\{M5, M4, M3\}$. After solving this new problem we exchange again the first unselected machine with the second selected machine, i.e., $M3$ with $M4$, and the machines selected in this case are $\{M5, M3, M2\}$. The following selection should be $\{M3, M4, M2\}$. At this point, where the first unselected machine has been exchanged with all machines selected in the first selection, we proceed to take the next unselected machine, $M1$. So, the next combination to try would be $\{M5, M4, M1\}$, then $\{M5, M1, M2\}$, and finally $\{M1, M4, M2\}$. If time permits after completing these selections, we continue with a random selection of machines.

3.3 "Not All Machines" algorithms

The first method to consider is the simple solution of the MIP mathematical model formed by the objective function (1) and constraint sets (2), (3), (4) and (5) with a modern commercial solver. We use the IBM ILOG CPLEX solver, in its last version 12.1 available at the time of the writing of this paper. We denote this solver as CPLEX in short.

A second straightforward method is to use CPLEX as a $R//C_{\max}$ solver, i.e., first carrying out the ranking and selection procedures and just using CPLEX to solve the unrelated parallel machines scheduling problem where some machines have been already removed. This means that each time we make a selection, the resulting reduced MIP mathematical model is solved with CPLEX. Since this reduced model is much smaller, it is expected to be solved much quicker. We set a maximum CPU time for each CPLEX run so that we can re-solve with a new set of machines provided by the machine selection procedure. The rationale behind stopping CPLEX before the current integer solution has been proven to be optimal is to avoid a possibly long span of time where CPLEX is just closing the search tree without improving results. Thus, a restart of CPLEX with a new selection of machines and providing the best result so far as a bound, allows CPLEX to usually find better solutions quickly. We refer to this second method as NAM+CPLEX

in short.

The third proposed algorithm also uses the machine ranking and selection procedures, but instead of using CPLEX as a solver for the resulting $R//C_{\max}$ problems, we use two heuristics. First we employ fast simple local search algorithm as a seed solution, which is later fed into a state-of-the-art method. The first heuristic is an insertion local search followed by an interchange local search, both iteratively applied in a loop until a local optimum is reached. This first heuristic is denoted as ST and interested readers can find complete explanations in the recent paper of **?**. Since this process is very fast, we can repeat ST a number of times (controlled by a maximum elapsed CPU time), each time with a different machine selection. This permits a fast heuristically found selection of good machines, along with a reasonable job-to-machine assignment.
The second heuristic is a more elaborate iterated greedy search method, called NVST-IG+ and proposed in **?**, which was later denoted in brief as DIG in **?**. Contrary to ST, DIG does not work over different machine selections. It is merely used the same way as CPLEX is used in NAM+CPLEX, i.e., only to work over the $R//C_{\max}$ problem.
This third method is denoted as NAM+ST+DIG. Notice that this method does not need any commercial solver for its application.

The fourth and last proposed method is NAM+ST+CPLEX. In this case, instead of launching CPLEX after the ranking and machine selection procedures, we first apply a fast local search in order to initialize CPLEX with a good seed solution. As we can see, all proposed methods are simple and easily reproducible.

3.4 Computational and statistical performance analysis

? proposed a comprehensive benchmark of no less than 1400 instances for the $R//C_{\max}$ problem. Instances are grouped into seven processing time distributions used to generate the processing times p_{ij} and, as the authors have shown, the different intervals have a profound effect on the results. All intervals employ discrete uniform distributions like intervals $U(1, 100)$, $U(10, 100)$, correlated jobs, correlated machines, $U(100, 200)$, $U(100, 120)$ and $U(1000, 1100)$. At each interval there are 10 instances for each combination of $n = \{100, 200, 500, 1000\}$ and $m = \{10, 20, 30, 40, 50\}$. Note the sheer size of the largest instances at 1000 jobs and 50 machines. In instances with correlated jobs, processing times are determined by the following expression: $p_{ij} = b_j + d_{ij}$ where b_j and d_{ij} are uniformly distributed values (also discrete) in the ranges $U(1, 100)$ and $U(1, 20)$, respectively. In the case of correlated machines, processing times obtained in a similar

way: $p_{ij} = a_i + c_{ij}$ where a_i and c_{ij} are uniformly distributed in $U(1, 100)$ and $U(1, 20)$, respectively.

We use a set of 12 PC/AT computers with Intel Core 2 Duo E6600 processors running at 2.4 GHz and 2 GB of RAM memory under the Windows XP SP3 operating system. No parallel processing is carried out with the 12 computers, we just simply divide the computational work over the 12 computers. Tests are conducted for different percentages of unused machines, i.e., for 20%, 50% and 80%. The stopping criteria for all methods is a maximum elapsed CPU time, which is accurately measured and has been set to 60 and 300 seconds. We have profusely used the Design of Experiments (DOE, **?**) methodology and the Analysis of Variance (ANOVA) statistical tool for drawing meaningful and sound conclusions. We check the three main hypotheses of the parametric ANOVA: normality, homoscedasticity and independence of the residuals. The tabulated results for each method will be presented as the relative percentage deviation from the best solution found as follows:

$$\text{Relative Percentage Deviation } (RPD) = \frac{C_{\max}(i) - C^*_{\max}(i)}{C^*_{\max}(i)} \cdot 100 \tag{8}$$

where $C^*_{\max}(i)$ is the aforementioned best solution found and $C_{\max}(i)$ is the value obtained by a given algorithm and instance i. All instances, together with the best solutions known are available at `http://soa.iti.es`. Note that comparing against the optimum solution is not viable since optimum solutions for the instances proposed could not be found in all cases. We will discuss later about comparisons against some optimum solutions or strong lower bounds.

Some of the proposed methods have some simple parameters that were calibrated. Basically, these are the times at which the different parts of the methods start and/or the maximum time allowed for each part. Table 3 shows this information and calibrated values according to the two elapsed CPU time stopping points.

Algorithm	Factor	Stopping time 60	300
NAM+CPLEX	Restart time for CPLEX	10	60
NAM+ST+DIG	Time given to ST	10	60
	Time given to DIG	50	240
NAM+ST+CPLEX	Time given to ST	20	100
	Time given to CPLEX	40	200
	Restart time for CPLEX	10	60

Table 3: Calibrated values for the different tested methods. Values in seconds.

We show the average results for each of the methods tested in the 1400 instances with the two stopping criteria and the three percentages of unused machines. Later, we show some statistical analyses of variance which represent the statistical significance of the observed differences between the various algorithms, their interactions and Tukey Honestly Significant Difference (HSD) confidence intervals with a 95% confidence level.
The results for 20% of unused machines, for elapsed CPU times stopping criteria of 60 and 300 seconds are reported in Tables 4 and 5, respectively. Similar tables, but for 50% and 80% of unused machines are reported in Tables 6 to 9.

Interval	CPLEX	NAM+CPLEX	NAM+ST+DIG	NAM+ST+CPLEX
$U(1,100)$	2.41	1.24	1.95	5.24
$U(10,100)$	2.10	0.75	0.58	2.89
Job Corre	1.76	0.66	0.21	1.29
Mach Corre	0.73	0.25	0.55	1.66
$U(100,200)$	0.84	0.30	0.17	0.69
$U(100,120)$	1.63	0.07	0.04	0.16
$U(1000,1100)$	0.10	0.04	0.02	0.08
Average	1.37	0.47	0.50	1.72

Table 4: Average relative percentage deviations for the "Not All Machines" algorithms with 20% of unused machines and 60 seconds elapsed CPU time stopping criterion.

Interval	CPLEX	NAM+CPLEX	NAM+ST+DIG	NAM+ST+CPLEX
$U(1, 100)$	0.54	0.69	1.51	5.00
$U(10, 100)$	0.68	0.18	0.39	2.78
Job Corre	0.69	0.31	0.06	1.02
Mach Corre	0.13	0.03	0.48	1.63
$U(100, 200)$	0.27	0.08	0.11	0.66
$U(100, 120)$	0.06	0.02	0.03	0.15
$U(1000, 1100)$	0.02	0.01	0.02	0.08
Average	0.34	0.19	0.37	1.62

Table 5: Average relative percentage deviations for the "Not All Machines" algorithms with 20% of unused machines and 300 seconds elapsed CPU time stopping criterion.

Interval	CPLEX	NAM+CPLEX	NAM+ST+DIG	NAM+ST+CPLEX
$U(1, 100)$	12.87	2.64	1.68	1.04
$U(10, 100)$	10.29	0.98	0.57	0.52
Job Corre	11.90	0.34	0.14	0.32
Mach Corre	3.73	0.29	0.74	0.20
$U(100, 200)$	12.88	0.33	0.27	0.26
$U(100, 120)$	14.20	0.07	0.06	0.07
$U(1000, 1100)$	1.94	0.04	0.02	0.03
Average	9.69	0.67	0.50	0.35

Table 6: Average relative percentage deviations for the "Not All Machines" algorithms with 50% of unused machines and 60 seconds elapsed CPU time stopping criterion.

Interval	CPLEX	NAM+CPLEX	NAM+ST+DIG	NAM+ST+CPLEX
$U(1, 100)$	5.11	1.05	1.68	0.34
$U(10, 100)$	2.94	0.49	0.41	0.10
Job Corre	3.65	0.16	0.05	0.32
Mach Corre	1.25	0.01	0.64	0.06
$U(100, 200)$	3.32	0.10	0.21	0.08
$U(100, 120)$	4.40	0.03	0.05	0.02
$U(1000, 1100)$	0.18	0.02	0.01	0.01
Average	2.98	0.27	0.44	0.13

Table 7: Average relative percentage deviations for the "Not All Machines" algorithms with 50% of unused machines and 300 seconds elapsed CPU time stopping criterion.

Interval	CPLEX	NAM+CPLEX	NAM+ST+DIG	NAM+ST+CPLEX
$U(1, 100)$	23.17	2.04	1.47	1.94
$U(10, 100)$	13.97	1.19	0.55	0.99
Job Corre	5.65	0.12	0.11	0.24
Mach Corre	56.64	0.04	0.44	1.33
$U(100, 200)$	7.72	0.24	0.15	0.38
$U(100, 120)$	7.23	0.07	0.07	0.14
$U(1000, 1100)$	5.47	0.04	0.03	0.06
Average	17.12	0.53	0.40	0.73

Table 8: Average relative percentage deviations for the "Not All Machines" algorithms with 80% of unused machines and 60 seconds elapsed CPU time stopping criterion.

Interval	CPLEX	NAM+CPLEX	NAM+ST+DIG	NAM+ST+CPLEX
$U(1, 100)$	12.95	0.67	0.63	1.19
$U(10, 100)$	7.90	0.47	0.29	0.70
Job Corre	2.45	0.07	0.06	0.19
Mach Corre	30.33	0.00	0.39	1.03
$U(100, 200)$	3.95	0.13	0.08	0.32
$U(100, 120)$	4.19	0.02	0.05	0.13
$U(1000, 1100)$	2.76	0.02	0.02	0.05
Average	9.22	0.20	0.22	0.52

Table 9: Average relative percentage deviations for the "Not All Machines" algorithms with 80% of unused machines and 300 seconds elapsed CPU time stopping criterion.

As we can see, the results of the proposed methods that employ our presented machine ranking and selection methods provide, for almost all cases of elapsed CPU time stopping criteria, percentage of unused machines and processing times intervals, significantly lower average relative percentage deviations from best known solutions. Some differences are striking, for example, we can see in Table 8 that solving the MIP mathematical model with CPLEX and stopping after 60 seconds of elapsed CPU time, and with 80% of unused machines, the last available version of CPLEX yields no less than a 23.17% average relative deviation, calculated for the 200 instances in the interval $U(1, 100)$. Comparatively, the proposed method NAM+ST+DIG, which does not use CPLEX at all, provides a meager 1.47% deviation from the best known solutions. Differences between CPLEX and the three proposed methods are much less marked when 300 seconds of elapsed CPU time are allowed. This is an expected result and, furthermore, with even more allowed CPU time,

eventually most methods would converge to the optimum solution.
It is interesting to observe that CPLEX behaves very well for just 20% of unused machines, beating two of the proposed methods for 300 seconds (Table 5). However, the results are much worse for 50% unused machines and specially, as commented, for 80% unused machines.
Table 10 presents the overall averages of all tested methods. Note that each cell contains the average results of 1400 instances. Among the presented methods, NAM+ST+CPLEX does not improve the results of NAM+CPLEX. Yet, under the situation where only 50% of the machines are considered – under both 60 and 300 seconds CPU time stopping criterion – NAM+ST+CPLEX is marginally better than NAM+CPLEX. However, NAM+CPLEX and NAM+ST+CPLEX improve the results of CPLEX in a significant way. This alone demonstrates that our presented machine ranking and selection procedures actually help when solving this interesting $R//C_{\max}$ problem generalization. Lastly, NAM+ST+DIG is the best method for 60 seconds and the second best for 300 seconds. In our humble opinion, this is a noteworthy result since NAM+ST+DIG does not make use of CPLEX.

	CPLEX	NAM+CPLEX	NAM+ST+DIG	NAM+ST+CPLEX
M.20% and 60 sec.	1.37	0.47	0.50	1.72
M.20% and 300 sec.	0.34	0.19	0.37	1.62
M.50% and 60 sec.	9.69	0.67	0.50	0.35
M.50% and 300 sec.	2.98	0.27	0.44	0.13
M.80% and 60 sec.	17.12	0.53	0.40	0.73
M.80% and 300 sec.	9.22	0.20	0.22	0.52
Average 60 sec.	9.39	0.56	0.47	0.93
Average 300 sec.	4.18	0.22	0.34	0.76

Table 10: Summary of relative percentage deviations from the best solutions know for the four "Not All Machines" methods tested in all intervals and stopping criteria.

As previously stated, all results are fed into a multifactor ANOVA where the different algorithms, intervals, percentage of unused machines, stopping time, number of machines and number of jobs are controlled factors. The response variable is the relative percentage deviation. Note that the total number of treatments is $1400 \cdot 6 \cdot 4 = 33,600$ (3,360 if the instance replicate is not studied as a witness factor) so the statistical power of the experiment is very high. Most significant factors have p-values that approach zero. Therefore, instead of comparing p-values, means plots with confidence intervals resulting from the ANOVA constitute a more practical approach. Some means plots are given in Figures 1 and 2.

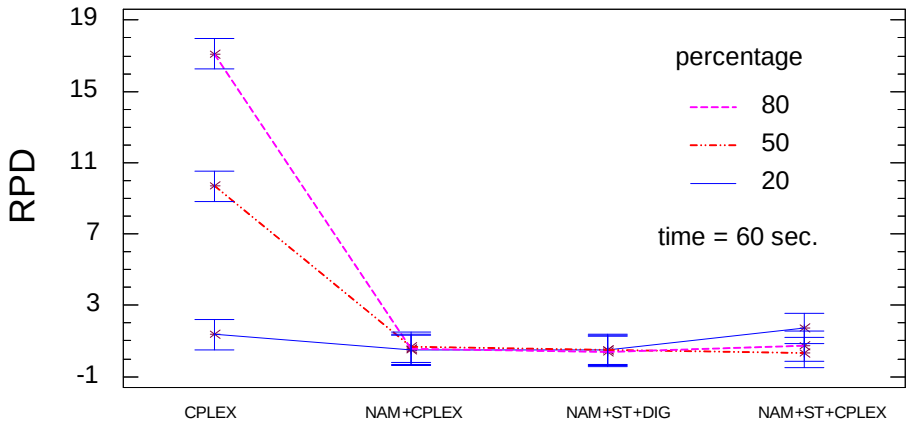

Figure 1: Relative percentage deviation means plot with Tukey HSD intervals at a 95% confidence level for the interaction between all four "Not All Machines" methods tested and all percentages of unused machines. 60 seconds elapsed CPU time stopping criterion.

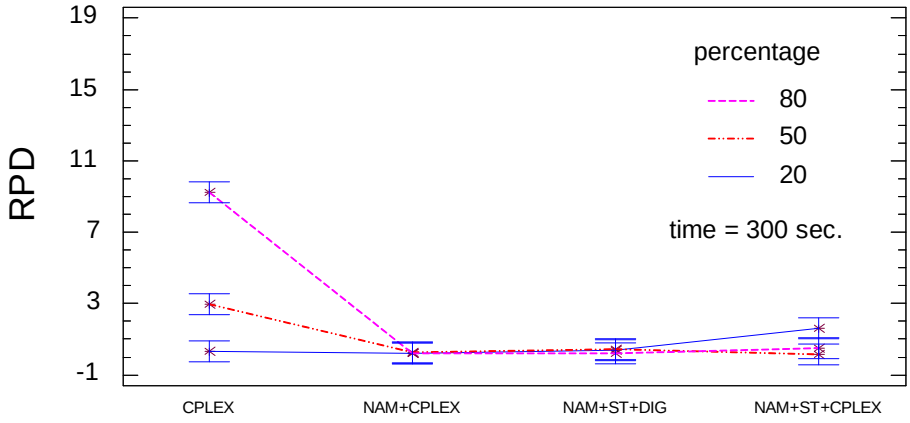

Figure 2: Relative percentage deviation means plot with Tukey HSD intervals at a 95% confidence level for the interaction between all four "Not All Machines" methods tested and all percentages of unused machines. 300 seconds elapsed CPU time stopping criterion.

As we can see, CPLEX produces average percentage deviations with differences that are statistically significant. Note that overlapping Tukey HSD intervals between two plotted means imply that the differences between the overlapped means are not statistically significant. In both cases, with 60 and 300 seconds elapsed CPU time stopping criterion both the percentage of unused machines and the algorithms factors, as well as the interaction between the two, resulted in p-values very close to zero. However, there are many non-statistically significant differences among the other tested methods. Statistical testing is, as we can see, necessary. From the overall averages given in Table 10, we conclude that much of the observed differences in average performance between NAM+CPLEX, NAM+ST+DIG and NAM+ST+CPLEX are not statistically significant.

NAM+ST+CPLEX is statistically worse than all other proposed methods for 300 seconds and 20% of unused machines. CPLEX is statistically equivalent to all other methods only for the case of 20% of unused machines. Although NAM+CPLEX and NAM+ST+DIG have similar results, we want focus on the fact than NAM+ST+DIG does not use any commercial solver. Commercial solvers are very expensive for industries and therefore, we prefer the simpler and "solver-less" NAM+ST+DIG method.

Comparing against the best known solution gives us relatively little information about the ultimate effectiveness of either CPLEX or the other tested methods. Comparing against true optimum solutions is a preferable option. However, the proposed MIP formulation, when solved with IBM ILOG CPLEX 12.1., is not able to solve all instances optimally, as the previous results have shown. In any case, we have carried out additional testing. Among all of our results with 20, 50 and 80% of unused machines and 300 seconds elapsed CPU time stopping criterion, we have calculated for how many instances the optimum solution could be obtained with CPLEX. Additionally, we also calculated for how many instances a gap of less than 1% between the lower bound and the best integer solution found is known (not including the previous optimally solved instances). Overall, there are 12.07% of instances with a known optimum and 7.67% of instances with a gap of less than 1%. We report the average relative deviations from these two sets of instances of the NAM+ST+DIG algorithm, run during 60 seconds in Table 11.

		20% unused mach gap < 1% optimum		50% unused mach gap < 1% optimum		80% unused mach gap < 1% optimum	
$U(1,100)$	% instances	8	31.5	1.5	12	0	15.5
	Average deviation	0.31	2.54	0.04	0.05	0.00	0.19
$U(10,100)$	% instances	9.5	10.5	1	9.5	0	15.5
	Average deviation	0.30	0.83	0.14	0.13	0.00	0.21
JobCorre	% instances	13.5	1	5.5	4.5	0.5	14.5
	Average deviation	0.33	0.05	0.21	0.26	0.27	0.22
MachCorre	% instances	37.5	39	16.5	26.5	3.5	19.5
	Average deviation	0.94	0.42	0.91	0.54	0.39	0.20
$U(100,200)$	% instances	14.5	1	8	2	1.5	13.5
	Average deviation	0.54	0.16	0.28	0.25	0.36	0.08
$U(100,120)$	% instances	3.5	12.5	6	4	1.5	10.5
	Average deviation	0.35	0.03	0.12	0.05	0.09	0.05
$U(1000,1100)$	% instances	16	0.5	10	0.5	3	9.5
	Average deviation	0.25	0.00	0.05	0.00	0.06	0.04
Total	% instances	14.64	13.71	6.93	8.43	1.43	14.07
	Average deviation	0.43	0.58	0.25	0.18	0.17	0.14

Table 11: Average relative percentage deviations of the NAM+ST+DIG algorithm run for 60 seconds with respect to optimum solutions or lower bounds for instances with a gap of less than 1% obtained using CPLEX 12.1 during 300 seconds. 20%, 50% and 80% of unused machines.

As shown, when comparing against the instances for which the optimum or very good lower bound is known, NAM+ST+DIG reports results of less than 0.29% average relative deviation (across all instances). This means that when the method works it works very well. However, there is a large percentage of instances for which no good lower bounds are known. Most presented methods, including CPLEX, provide solutions that are not as far apart as the gap values indicate. Therefore, all points out to a poor lower bound inside CPLEX (which basically depends on the linear relaxation of the solved MIP model). We already observe from Table 11 that increasing the percentage of unused machines decreases the number of optimal solutions found by CPLEX. The way processing times are distributed also affects the optimality rate. For example, for correlated machines and 20% unused machines, the optimum is known for 39% of the instances. Comparatively, only a 0.5% of the $U(1000,1100)$ instances have a known optimum. Apart from that, a close analysis of all the experimental data did not yield any further interesting conclusions as regards which factors affect the large gaps.

4 The "Not All Jobs" (NAJ) problem

As with the previous NAM $R//C_{\max}$ generalization, the first step is to test the MIP mathematical model composed of the objective function (1), followed by constraint sets (2), (6) and (7). This MIP model is solved as a first step with CPLEX with a stopping elapsed CPU time of 300 seconds with the condition that just 50% of jobs must be processed. The results are shown in Table 12. Each cell represents the average of the 200 instances of each processing time interval. However, in this case the relative deviation has been calculated with respect to the lower bound given by CPLEX at the time limit of 300 seconds or with respect to the optimum solution whenever CPLEX was able to solve instances optimally. We also present the average gap between the reported solution and the mentioned lower bound. The maximum deviation is also given.

Interval	Maximum deviation	Average deviation	Average gap
$U(1,100)$	0.00	0.00	0.00
$U(10,100)$	2.26	0.17	1.28
Job Corre	12.28	1.94	2.36
Mach Corre	1.13	0.01	0.48
$U(100,200)$	3.28	0.11	0.49
$U(100,120)$	0.08	0.00	0.05
$U(1000,1100)$	8.34	1.01	1.87
Average	3.91	0.46	0.93

Table 12: Maximum and average deviations from the lower bound or optimum solution, together with the gap for CPLEX MIP mathematical model solution for the "Not All Jobs" problem. 50% of jobs to be processed and 300 seconds elapsed CPU time stopping criterion. All results in percentages.

It can be seen that CPLEX alone provides very good values which are, in average, below 0.5%. Some intervals, like $U(1,100)$ result in the optimum solution for all 200 instances tested. However, for some other intervals, like correlated jobs, we observe a maximum deviation of 12.28% but still the average deviations, and above all, average gap, are very small. This last result is not surprising, when jobs are correlated (some jobs are faster and some others are slower on all machines) it is more difficult to decide which jobs to eliminate. The slower jobs are easy to rule out but once all slow jobs have been eliminated, a hair splitting process is needed to finish off with a 50% processed jobs because of the inherent relationship between jobs. However, the average deviation in this case is still below 2%.

We also tested the model where 20% and 80% of jobs were not processed. For 20% of

non-processed jobs, the average deviations from the lower bound or optimum solution was a bit higher at 0.51%. For 80% of non-processed jobs the results were even lower at just 0.33%. Furthermore, for 80% of non-processed jobs, 5 out of the 7 tested intervals resulted in optimum solutions across all instance sizes when run for 300 seconds of CPU time. Given these results it seems unnecessary to propose specific algorithms for this problem, given the excellent performance observed with CPLEX, specially if one considers that only five minutes of CPU time have been allotted.

Despite these good results, we still attempted several adaptations and algorithms. We employed job selection methods, similar to the previous machine ranking and selection procedures, with the objective of ranking and selecting jobs. In a similar way, we apply the ST and DIG heuristics to obtain an algorithm that we have referred to as NAJ+ST+DIG. These algorithms solve the unrelated parallel machines problem just with the jobs selected after the rankings. The results were not competitive with those obtained by CPLEX. As an example we show in Figures 3, 4 and 5 the means plot resulting from an ANOVA analysis for three different intervals between CPLEX and the proposed NAJ+ST+DIG method. It is clear that CPLEX is statistically better than the proposed algorithm adapted to this problem, except in the case of correlated jobs, where despite not being statistically different, CPLEX still has a better average. This outcome is expected as we anticipated, given the very good results obtained with CPLEX.

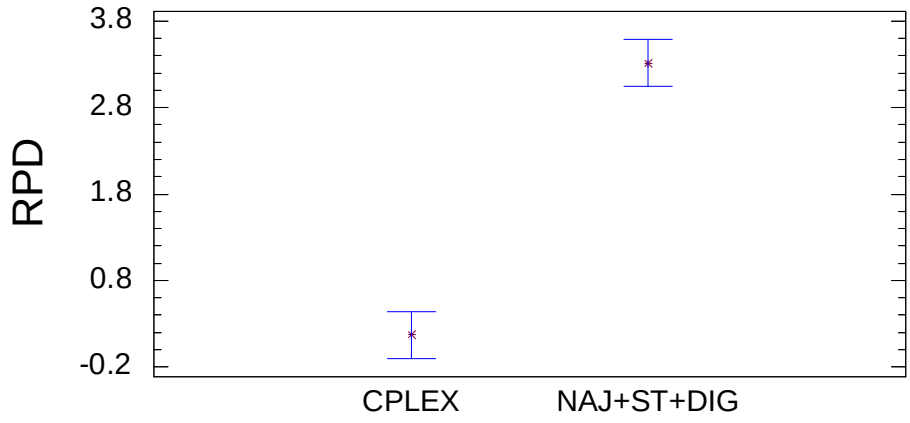

Figure 3: Means plot and Tukey HSD intervals at a 95% confidence level with average relative deviations from lower bounds or optimum solutions of CPLEX and NAJ+ST+DIG. 50% of jobs to be processed and 300 seconds elapsed CPU time stopping criterion for processing time interval $U(10, 100)$.

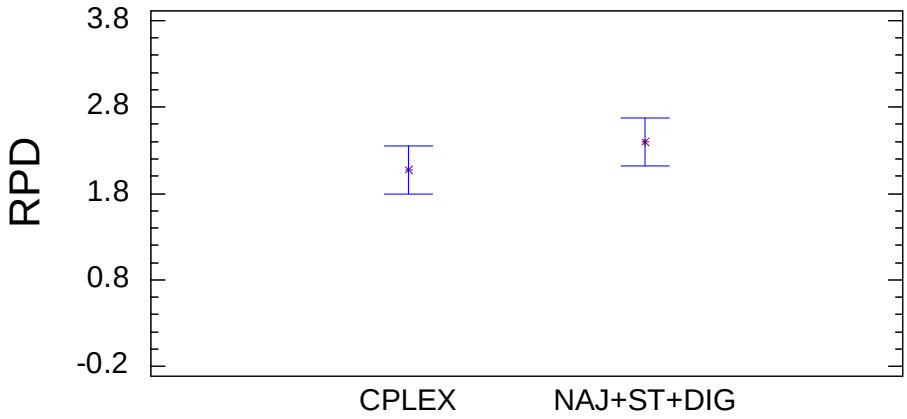

Figure 4: Means plot and Tukey HSD intervals at a 95% confidence level with average relative deviations from lower bounds or optimum solutions of CPLEX and NAJ+ST+DIG. 50% of jobs to be processed and 300 seconds elapsed CPU time stopping criterion for processing time interval of correlated jobs.

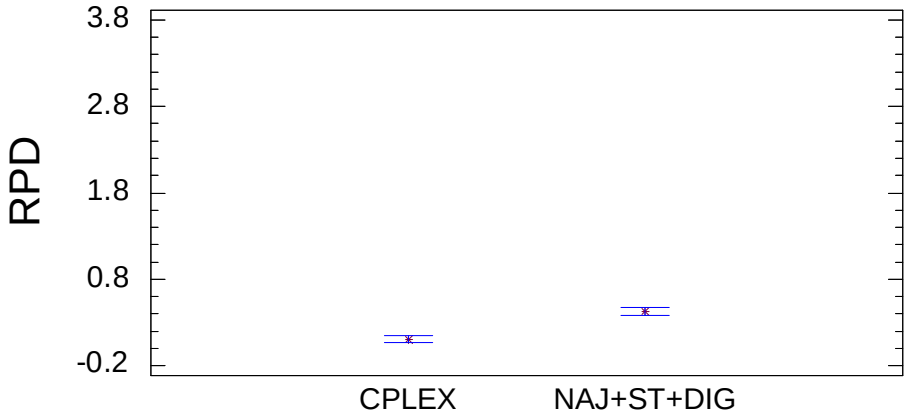

Figure 5: Means plot and Tukey HSD intervals at a 95% confidence level with average relative deviations from lower bounds or optimum solutions of CPLEX and NAJ+ST+DIG. 50% of jobs to be processed and 300 seconds elapsed CPU time stopping criterion for processing time interval $U(100, 200)$.

5 Conclusions and future research

In this paper we have studied, for the first time, and to the best of our knowledge, two generalizations of the problem of sequencing jobs on unrelated parallel machines with the objective of minimizing the $C_{\max}$. These generalizations are the result of including more tactical or strategic decisions in the scheduling process. More specifically, we have studied first the problem in which not all available parallel machines need to be used and the additional decision is to determine which machines should be ruled out. The second studied

generalization affects jobs and the supplemental action is to decide which jobs must be processed. Mixed Integer Programming (MIP) mathematical models have been presented, along with some additional methods specifically tailored for such generalizations.

For the "Not All Machines" (NAM) problem we have presented a very simple machine ranking procedure that sorts machines from most promising to less interesting as regards $C_{\max}$ minimization. Together with the ranking, we have also devised an equally simple machine selection procedure that selects machines in a smart way using the previous ranking. These two simple procedures have been coupled with either CPLEX or recent state-of-the-art algorithms that have been proven to be very effective when solving the unrelated parallel machine scheduling problem. Comprehensive computational and statistical analyses, carried out over a wide range of 1400 instances, with different parameters and stopping time criteria allow us to conclude that the presented machine ranking and selection procedures provide solutions that are many times better than those produced with CPLEX.

Conversely, CPLEX provides very good solutions for the second studied generalization of "Not all Jobs" or NAJ. The simple MIP mathematical model is solved by CPLEX to almost optimality with average deviations from lower bounds below 0.5% in under five minutes of CPU time.

In our opinion, many possible further studies stem from the NAM and NAJ generalizations. First of all, these problems can be naturally extended to a multi-objective setting as wildly different $C_{\max}$ values are to be expectedly obtained for any number of used machines. Therefore, two objectives, namely, $C_{\max}$ and number of machines used, can be simultaneously optimized. Makespan is hardly the only possible scheduling objective and other even more interesting results could be obtained by studying due date satisfaction together with number of machines. NAJ multi-objective settings seem equally interesting as well.
Additionally, we have presented in this paper markedly simple methods. Still, solutions could be improved by using more elaborate methods and/or advanced exact methodologies.
Finally, parallel machine problems are not the only scheduling settings where these NAM and NAJ generalizations can be applied. NAJ can be actively applied to interesting single machine problem variants as those presented in **?** or **?**, just to name two recent examples. NAM can be applied to every stage of hybrid flowshops that are now being actively

researched like in ? or as the recent review paper of ? shows.

Acknowledgments

The authors would like to thank the anonymous referees for their careful and detailed comments which have helped improve this manuscript considerably. 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, under the project "OSC" with references IMIDIC/2008/137, IMIDIC/2009/198 and IMIDIC/2010/175 and the Polytechnic University of Valencia, under the project "PPAR - Production Programming in Highly Constrained Environments: New Algorithms and Computational Advances" with reference 3147.