Hybrid tabu search and a truncated branch-and-bound for the unrelated parallel machine scheduling problem

Veronique Sels[1], José Coelho[2], António Manuel Dias[2], and Mario Vanhoucke[1,3,4]

[1]Faculty of Economics and Business Administration, Ghent University, Tweekerkenstraat 2, 9000 Gent (Belgium), veronique.sels@ugent.be, mario.vanhoucke@ugent.be

[2]Universidade Aberta, Rua da Escola Politécnica, 147, 1269-001, Lisbon (Portugal), jcoelho@uab.pt

[3]Operations and Technology Management Centre, Vlerick Leuven Gent Management School, Reep 1, 9000 Gent (Belgium)

[4]Department of Management Science and Innovation, University College London, Gower Street, London WC1E 6BT (United Kingdom)

Abstract

We consider the problem of scheduling a number of jobs on a number of unrelated parallel machines in order to minimize the makespan. We develop three heuristic approaches, i.e., a genetic algorithm, a tabu search algorithm and a hybridization of these heuristics with a truncated branch-and-bound procedure. This hybridization is made in order to accelerate the search process to near-optimal solutions. The branch-and-bound procedure will check whether the solutions obtained by the meta-heuristics can be scheduled within a tight upper bound. We compare the performances of these heuristics on a standard data set available in the literature. Moreover, the influence of the different heuristic parameters is examined as well. The computational experiments reveal that the hybrid heuristics are able to compete with the best known results from the literature.

1 Introduction

The parallel machine scheduling problem consists of scheduling a set of jobs N on a set of parallel machines M without interruption. Each job j (index j = 1, ..., n) becomes available for processing at time zero and has to be processed by exactly one out of the m (index i = 1, ..., m) parallel machines. Every machine can process only one job at a time and once a job is started on a certain machine, no preemption is permitted. In general, various classes of parallel machine scheduling problems have been defined to minimize the maximal completion time (C_i) of the machines or the makespan ($C_{max} = \max_i C_i$), depending on how the processing times of the jobs are defined. In the case where the processing time p_j of a job j is independent of the machine, the problem is known as the *identical* parallel machine scheduling problem. The problem can be represented by $P||C_{\max}$ using the classification scheme of Graham et al. (1979). In the case where each machine i has a different speed s_i for processing the jobs, it is referred to as the *uniform* parallel machine scheduling problem ($U||C_{\max}$) and the processing time of a job j on machine i is derived as follows: $p_{ij} = p_j/s_i$. When the processing time p_{ij} depends on the machine on which the job j is scheduled, but no relationship exists between the machine speeds, the problem is known as the *unrelated* parallel machine scheduling problem ($R||C_{\max}$). In this paper, we consider the unrelated parallel machines scheduling (UPMS) problem with the objective of minimizing the maximal completion time of the machines, which is known to be NP-hard (Garey and Johnson, 1979).

The literature on (identical) parallel machine scheduling is very extended. However, the unrelated parallel machine scheduling problem has been much less studied. In the following paragraphs, an overview is given of the most relevant papers that focus on the unrelated parallel machine problem with a makespan objective. With respect to exact approaches, we mention the work of van de Velde (1993) and Martello et al. (1997). van de Velde (1993) proposes an exact algorithm using a branch and bound technique and an iterated local search meta-heuristic, which are both based on a dual approach. The paper of Martello et al. (1997) discusses effective lower bounds for the problem based on Lagrangian relaxation and presents some approximate algorithms. Some efficient heuristic approaches are discussed in Glass et al. (1994), Piersma and Van Dijk (1996), Srivastava (1998), Sourd (2001), Mokotoff and Chrétienne (2002), Mokotoff and Jimeno (2002), Ghirardi and Potts (2005), Monien and Woclaw (2006), Guo et al. (2007), Fanjul-Peyro and Ruiz (2010), Fanjul-Peyro and Ruiz (2011) and Lin et al. (2011). Glass et al. (1994) perform a comparison between a genetic algorithm, a simulated annealing approach and a tabu search algorithm. Another effective tabu search algorithm is presented in the paper of Srivastava (1998). Guo et al. (2007) develop hybrid heuristic methods based on simulated annealing, tabu search and squeaky wheel optimization. Piersma and Van Dijk (1996) propose a local search technique with an efficient neighborhood search. Similarly, Sourd (2001) presents two heuristic algorithms based on a large neighborhood search. Monien and

Woclaw (2006) present an experimental study on the unsplittable Truemper algorithm and Lin et al. (2011) discuss a two-phase scheduling heuristic with two improvement methods. Mokotoff and Chrétienne (2002) develop a cutting plane algorithm that deals with the polyhedral structure of this UPMS problem. This algorithm is further refined in Mokotoff and Jimeno (2002), where heuristic algorithms based on partial enumeration are presented. This approach was considered to be the most effective approach for $R||C_{max}$, together with the recovering beam search of Ghirardi and Potts (2005). However, Fanjul-Peyro and Ruiz (2010) present a set of iterated greedy local search based heuristics that outperform the aforementioned state-of-the-art procedures. In addition, Fanjul-Peyro and Ruiz (2011) propose meta-heuristics based on a size reduction of the original job-machine assignment problem with which they challenge the well-performing methods of Fanjul-Peyro and Ruiz (2010). They prove that they can obtain better results by running their algorithms in parallel mode having several physical cores or processors available and keeping the best result at the end.

The contribution of this paper can be summarized as follows. First, we develop three meta-heuristic approaches to solve the UPMS problem. These meta-heuristics are improved by the inclusion of a local search algorithm that takes problem specific information into account. Second, we present a hybridization of these meta-heuristics with a novel truncated branch-and-bound procedure in order to accelerate the search process to near-optimal solutions. As such, the meta-heuristics are guided in their search process to skip non-promising areas of the solution space. Third, we compare the performances of these heuristics on a standard data set available in the literature. We carefully examine the influence of the different heuristic parameters, such as the backtrack limit and the upper bound strategy, to select the best combination for the hybrid algorithms and check the influence of the problem parameters on the performance of these hybrid meta-heuristics.

The outline of this paper can be summarized as follows. Section 2 gives a brief overview of the stand-alone genetic algorithm, tabu search procedure and the local improvement technique developed for the problem under study. Section 3 describes the principles and the implementation of the truncated branch-and-bound procedure. In Section 4, the computational experiments are discussed, which show the promising results of the hybrid heuristics in solving the unrelated parallel machine scheduling problem. Section 5 summarizes the main conclusions and contributions.

2 Meta-heuristic solution procedures

2.1 Solution representation

A commonly used representation technique for the parallel machine scheduling problem is described in Cheng and Gen (1997) and Mendes et al. (2002). This representation consists of a permutation encoding with job symbols and partitioning symbols, where every chromosome is a permutation of the n jobs in which each job appears exactly once. $m-1$ partitioning symbols (*) are used to represent the assignment of the jobs to the different machines. As such, the solutions are represented by a string with size $n+m-1$.

To illustrate this, we give an example of an unrelated parallel machine scheduling problem with 6 jobs and 3 machines in Table 1. The table shows the processing times of jobs on the different machines. A possible solution for this problem can be represented by the string: 4 5 * 1 3 * 2 6. According to this solution, machine 1 will process jobs 4 and 5 in that sequence, machine 2 will execute jobs 1 and 3 and machine 3 will process jobs 2 and 6, resulting in a makespan value of 9, which is equal to the maximal completion time obtained by machine 3 (i.e., $p_{32}+p_{36}=5+4=9$).

Table 1: Example of an UPMS instance with 10 jobs and 3 machines

	Machines		
Jobs	i=1	i=2	i=3
j=1	9	3	6
j=2	7	8	5
j=3	6	5	10
j=4	3	4	7
j=5	2	5	6
j=6	6	7	4
	Processing times		

2.2 Tabu Search algorithm

2.2.1 General

The tabu search algorithm (TS), introduced by Glover (Glover and Laguna, 1997), is a well-known meta-heuristic for finding near optimal solutions in combinatorial optimization problems. The TS can be seen as a hill-climbing technique that tries to prevent local optimality by allowing non-improving moves. The principles of the TS can be briefly described as follows (Gendreau and Potvin, 2010). The algorithm starts with a feasible initial solution and tries to iteratively improve the solution by examining a set of candidate moves in the neighborhood of that solution. The neighborhood contains all feasible solutions that can be obtained from the current solution by executing a simple move. The neighboring solutions are evaluated based on their objective function value and the current solution moves to its best neighboring solution, even if it results in a worse objective value. The best move is put on the tabu list, so that it cannot be made undone for some number of iterations in order to avoid a cyclic search. The number of iterations that a move is set tabu is determined by the size of the tabu list. The size of the list must be large enough to prevent cycling and to emphasize exploration and small enough to ensure the intensification of the search. Furthermore, an aspiration criterion is defined to deal with the situation where the moves are tabu. A commonly used aspiration criterion allows a tabu move when it results in a solution with an objective value better than the current best-known solution. The algorithm stops when a certain stopping criterion is satisfied and the best found solution so far is returned as the final solution.

2.2.2 Procedure

Our TS algorithm starts with the initial solution obtained by the shortest processing time (SPT) rule. With this rule, every job is assigned to the machine on which it has the smallest processing time. The initial solution is then inserted in the tabu search procedure. The tabu search makes use of the general pairwise swap (GPS) neighborhood (Laguna et al., 1991), in which every job and/or partitioning symbol is swapped with every other job and/or partitioning symbol. Swapping two jobs that belong to different machines results in different job-machine assignments for both jobs, swapping a job with a partitioning symbol can results in a completely different schedule. For example, if we swap job symbol 5 with job symbol 2 in the solution 5 3 * 2 4 * 1 6 described above, this results in job 5 scheduled on machine 2 and job 2 scheduled on machine 1. However, if we swap job symbol 5 with the first partitioning symbol in the sequence, this leads to the solution where no jobs are scheduled on machine 1 and 4 jobs scheduled on machine 2 (i.e., * 3 5 2 4 * 1 6). Every swap is evaluated to determine its influence on the machines' completion times. The best possible swap that results in the smallest makespan value is then executed, unless the swap is prohibited by the tabu list. However, if a tabu swap results in a makespan that is less than the overall minimum obtained so far, the swap is accepted nevertheless (aspiration criterion). The tabu list is then updated with the best swap. The tabu list is a matrix with dimension of $(n+m-1)\times(n+m-1)$. The value in the tabu list matrix is set equal to the current iteration plus the tabu list size, which is equal to the number of next iterations the (reverse) swap will be tabu. After the execution of the best swap, the current solution is further improved by a local search algorithm, which will try to improve the solution by moving jobs to other machines. Because of the importance of the local improvement technique, we discuss it in more detail in Section 2.5. The new solution is evaluated and the search process continues with this newly obtained solution. The tabu search stops after a specified number of generated schedules and returns the best solution found.

Because of the GPS neighborhood, the maximum number of possible moves is equal to $\sum_{i=1}^{(n+m-1)-1}((n+m-1)-i)$ and thus the number of neighboring solutions will increase as the number of jobs increases. Due to this large neighborhood size and the resulting computational burden, a modification can be be made to restrict the neighborhood as much as possible without compromising the efficiency of the TS algorithm (Shin et al., 2002). For that reason, we have examined a number of ways to restrict the general pairwise swap neighborhood in a conditional manner. The first restricted neighborhood (NH) requires that the jobs to be swapped belong to different machines. The second NH swaps jobs on different machines if at least one of them has a different processing time on the other machine. The third NH only considers swaps of jobs if they result in an improvement of the sum of the processing times. The fourth NH restricts the neighborhood by requiring that one of the corresponding machines is the critical machine (i.e., machine with highest completion time), but relaxes the requirement of the improvement of the third NH. The fifth NH is a combination of NH3 and NHY4 and restricts the neighborhood by requiring an improvement in the sum of the processing times and that one of the machines is critical. The selection of the best neighbourhood restriction is done on a small dataset and will be discussed in Section 4.2.2.

2.3 Genetic algorithm

2.3.1 General

The genetic algorithm (GA) is a population-based meta-heuristic based on the principles of genetics and natural selection. The method was initially developed by John Holland in the 1970s (Holland, 1975), who was inspired by Darwin's evolution theory *"the survival of the fittest"*. A GA starts with a population of (randomly) generated solutions, which is improved iteratively. A single iteration is referred to as a generation. The population of one generation consists of individuals surviving from the previous generation plus the newly generated solutions, while keeping the population size constant. Each individual of the population is characterized by its fitness value. This fitness value is related to the associated value of the objective function. Better solutions of the population are selected to serve as parents in a recombination phase. The recombination phase uses a crossover operator to combine the characteristics of the parents in the creation of new solutions and a mutation operator to alter these new solutions to ensure an extensive search of the solution space. The obtained solutions enter the population of the next generation replacing the worst solutions of the current generation. This new population is evaluated and recombined again until a certain stopping criterion is met.

2.3.2 Procedure

In contrast to the tabu search, the GA starts with a population of randomly generated solutions. However, we seed this population with the SPT solution in order to improve the quality of the future solutions and to speed up the search process (Reeves, 2003). In order to determine the best combination of reproduction operators, such as selection, crossover and mutation operators, we have performed a full factorial design where we tested every possible combination of 3 selection methods (i.e., tournament selection, roulette wheel selection and ranking selection), 5 crossover operators (i.e., partially mapped crossover, order crossover, position based crossover, order based crossover and cycle crossover) and 3 mutation operators (i.e., swap, insertion and inversion mutation), resulting in $3 \times 5 \times 3 = 45$ possible combinations.

Based on the results of this computational experiment, we make use of the tournament selection method, where we compare pairs of solutions and select the best one as being the parent solution (based on their makespan values). The selected solutions are then recombined by the order crossover (OX) operator, which is executed with a certain probability, the crossover rate. In the OX, two crossover points are selected at random from the first parent solution and the job and/or partioning symbols between these positions are copied to the same positions of new solution. Those symbols are deleted from the other parent solution and the remaining symbols are inserted into the new solution according to the order they appear in the second parent solution (Kellegöz et al., 2008). In order to ensure some diversity in the population, these solutions are then mutated with the swap mutation, where a randomly chosen job and/or partitioning symbol is swapped with another random symbol in the sequence. Mutation is also executed with a certain, rather low, probability, the mutation rate. Before the new solutions are introduced into the population, they are improved by the best local search described in Section 2.5. The solutions are evaluated and inserted back in the population by using an incremental replacement approach. The new (unique) solutions will replace the solutions with the worst makespan to ensure that the best solutions will survive to the next generation. The GA is stopped when a limit on the examined schedules is exceeded.

2.4 Sampling heuristic

In order to validate the contribution of the meta-heuristic solution approaches, we have also implemented a sampling heuristic. This sampling heuristic generates a number of random solutions that are improved by the local improvement technique described in Section 2.5. This process is repeated for a maximum number of iterations and the best-found solution is returned as the final solution.

A graphical representation of the genetic algorithm, the tabu search procedure and the sampling heuristic is given in Figure 1. Moreover, we will examine the hybridization of these meta-heuristics with a truncated branch-and-bound procedure in order to accelerate the search process to near- optimal solutions. This hybridization will takes place in the evaluation phase after each iteration of the meta-heuristic, when newly obtained schedules are evaluated on their makespan values (labelled as 'Evaluation' in Figure 1). As a result, the branch-and-bound algorithm will serve as a guidance tool in the meta-heuristic search process.

2.5 Local improvement technique

The local search algorithm that we have implemented consists of two neighborhoods, i.e., an insertion and a swap neighborhood, which are examined consecutively. In the insertion neighborhood, the move of jobs to other machines is examined, while in the swap neighborhood, we consider the swapping of jobs belonging to different machines. We have implemented and tested four different approaches, which differ in the job and machine selection strategy. These approaches are, to some extent, based on the variable neighborhood descent method as proposed by Fanjul-Peyro and Ruiz (2010). The first approach ($LS1$)

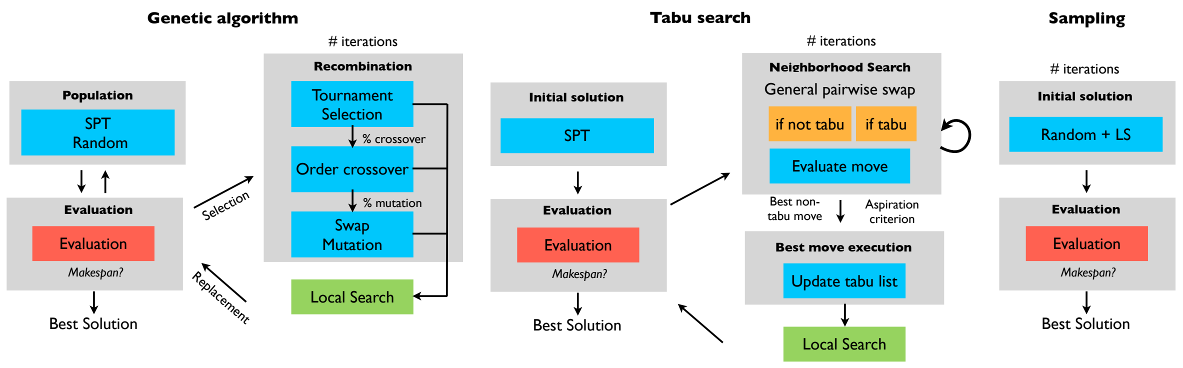


Figure 1: General outline of the genetic algorithm (left), tabu search (middle) and sampling heuristic (right)

focuses on the critical machine h, i.e., the machine responsible for the maximal completion time or makespan value ($C_h = C_{max}$). In the first phase, we try to relocate each job j belonging to this critical machine to another machine $i \in M \setminus \{h\}$, where the scheduling of this job results in the lowest increase in completion time of that machine. The move is accepted if the new completion time $C_i + p_{ij}$ is less than the original makespan value C_{max}. This process is repeated until no more improvement can be found or until a time limit has been exceeded. In the second phase, we try to swap every job j of the critical machine h, which can differ from the original critical machine due to changes made in phase 1, with every other possible job k located at another machine $i \in M \setminus \{h\}$. Only swaps that result in a net gain of processing times, i.e., the sum of the new processing times is less than the sum of the old processing times ($p_{hj} + p_{ik} < p_{ij} + p_{hk}$), are taken into consideration. The swap that results in the lowest increase in completion time on the new machine i is accepted if the new completion times of the critical and new machine are less than the original makespan value C_{max}. Again, phase 2 is repeated until no more improvement can be found or when the time limit is exceeded.

The second approach ($LS2$) also selects jobs to be moved or swapped from the critical machine h, but differs from the first approach in the condition of rescheduling. In phase 1, every job j of the critical machine is assigned to another machine $i \in M \setminus \{h\}$., such that the scheduling of the job results in the lowest increase in processing time, i.e., the difference between its original processing time p_{hj} (on the critical machine) and its new processing time p_{ij} (on the chosen machine i). In the second phase, this condition translates to the smallest difference in the sum of the new and old processing times $((p_{ij} + p_{hk}) - (p_{hj} + p_{ik}))$.

The moves and/or swaps are accepted if the newly obtained completion times are less than the original makespan value. Similar to LS1, both phases are repeated until no more improvement can be found or when a time limit has been exceeded.

The third ($LS3$) and fourth ($LS4$) approach are similar to the first two approaches, except that they are not restricted to the critical machine, but allow moves and/or swaps of jobs from every machine $i \in M$. The third approach is exactly the approach described in the paper of Fanjul-Peyro and Ruiz (2010), $LS1$ is a restricted version of this approach, while $LS2$ and $LS4$ are newly developed approaches.

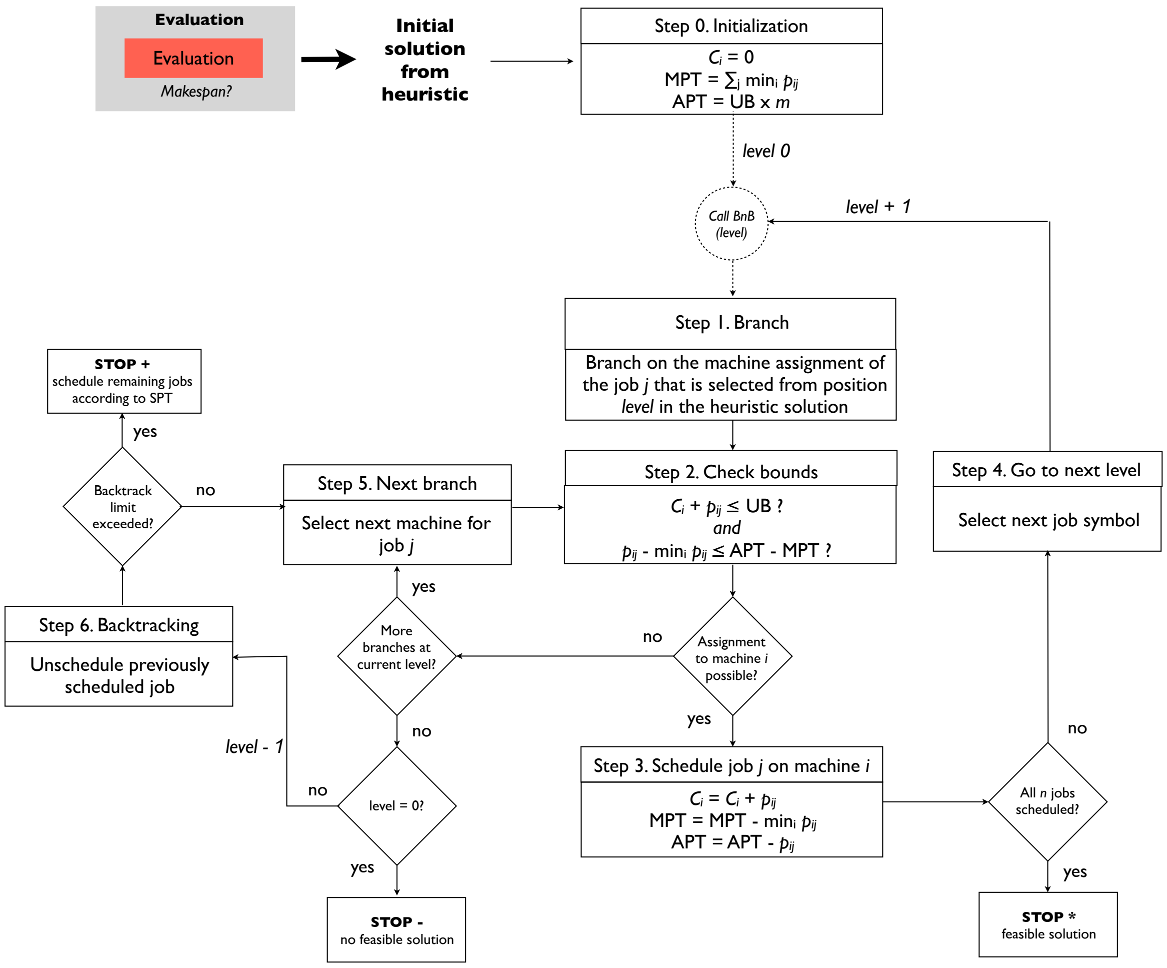
These four local search approaches are tested in Section 4.2.1, to determine the best approach for implementation in the meta-heuristics.

3 Hybridization with truncated branch-and-bound

In this Section, we describe the hybridization of our meta-heuristics and the sampling heuristic with a branch-and-bound procedure. As we have already mentioned in the previous sections, the branch-and-bound algorithm will evaluate each solution resulting from the heuristics by means of an upper bound set on the makespan value. More precisely, the algorithm will check the possibility of scheduling the corresponding solution, with the jobs assigned to a certain machine, within that upper bound. The general principles of the branch-and-bound procedure will be described in Section 3.1. The specific implementation details, with the discussion of the upper and lower bound strategy and the truncation, will be discussed in Section 3.2. In addition, in order to illustrate these general principles and implementation details, an overview of the algorithm is given in Figure 2.

3.1 Principles

The general procedure of the branch-and-bound can be summarized as follows. After initialization (*Step 0*), the algorithm examines the solution obtained by the heuristic and considers the job symbols successively in the order in which they appear in the list (*Step 1*). The procedure will check whether the assignment of the job to the current machine is possible by comparing the resulting machine completion time with an upper bound (*Step 2*). If this assignment is possible within the upper bound (i.e., resulting makespan $\leq$ UB), the job is scheduled on the corresponding machine and the branch-and-bound is recursively called to go to the next level in the search tree (*Step 3*). The next level corresponds with the next job symbol in the solution (*Step 4*), for which the same examination is made. However,



Evaluation
Evaluation
Makespan?
Initial solution from heuristic
Step 0. Initialization
$C_i = 0$
$\text{MPT} = \sum_j \min_i p_{ij}$
$\text{APT} = \text{UB} \times m$
level 0
Call BnB (level)
level + 1
Step 1. Branch
Branch on the machine assignment of the job j that is selected from position level in the heuristic solution
Step 2. Check bounds
$C_i + p_{ij} \leq \text{UB}$? and $p_{ij} - \min_i p_{ij} \leq \text{APT} - \text{MPT}$?
Step 4. Go to next level
Select next job symbol
Step 5. Next branch
Select next machine for job j
STOP + schedule remaining jobs according to SPT
yes
Backtrack limit exceeded?
no
Step 6. Backtracking
Unschedule previously scheduled job
yes
More branches at current level?
no
Assignment to machine i possible?
yes
Step 3. Schedule job j on machine i
$C_i = C_i + p_{ij}$
$\text{MPT} = \text{MPT} - \min_i p_{ij}$
$\text{APT} = \text{APT} - p_{ij}$
no
All n jobs scheduled?
yes
STOP * feasible solution
no
level - 1
no
level = 0?
yes
STOP - no feasible solution

Figure 2: Flow chart of the truncated branch-and-bound procedure

if a certain job-machine assignment is not possible with respect to the imposed upper bound, the procedure will try to assign the current job to one the other machines available. The order in which these other machines are examined is determined by the shortest processing time first-rule (*Step 5*). This means that the next machine to be considered is the one on which the job has the smallest processing time. If another possible assignment has been found, the job is scheduled on the corresponding machine and the branch-and-bound continues with the next level in the search tree. However, if all machine assignments of a particular job have been checked and none of these assignments are possible with respect to the upper bound, the branch-and-bound will return an infeasible solution and backtracking is needed (*Step 6*). This backtracking will unschedule previously scheduled jobs and will examine another machine assignment for those jobs that can resolve the conflict found. If a new possible assignment is found that resolves the upper bound violation, the branch-and-bound continues from the last rescheduled job. However, if the upper bound violation can not be resolved after backtracking (and before the backtrack limit is achieved, see infra), no feasible solution can be found and the branch-and-bound returns the same solution back to the (meta-)heuristic, which will try to improve it further (*STOP -*). However, if a complete feasible solution has been found, this solution can result in other job-machine assignments as obtained by the (meta-)heuristic (*STOP **). This new solution will only be accepted if the newly obtained makespan value is less than the original makespan value. This will speed up the search process of the meta-heuristics, as this improved solution will be the input again of the meta-heuristics. As a result, while the heuristics search in a restricted neighborhood, the branch-and-bound tries to jump out of the restricted neighborhood.

After the branch-and-bound procedure, the upper bound is updated according to a certain strategy. Moreover, the branching process is guided by means of an effective lower bound. In addition, to increase its effectiveness, the branch-and-bound procedure is truncated by restricting the number of backtrack limits. These implementation details will be discussed in the next section.

3.2 Implementation details

Truncation Since the branch-and-bound is hybridized with (meta-)heuristic algorithms, the procedure has to be fast and efficient in returning a feasible solution. For that reason we have truncated the branch-and-bound by defining a maximum number of backtrack limits. This limit will restrict the number of times we unschedule a job in order to resolve the upper bound violations. If the backtrack limit is attained, the remaining unscheduled jobs are scheduled according to the SPT-rule, by assigning them to the machine on which they have the smallest processing time (*STOP +*). As such, the branch-and-bound will return a feasible heuristic solution, but with a makespan higher than the upper bound. In Section 4.3, various backtrack limits will be compared in order to find the best value for the hybrid algorithms.

Lower bound The branch-and-bound is guided by a lower bound. The initial lower bound is equal to the makespan value obtained by the SPT-rule divided by the number of machines. However, this rather simple lower bound is updated during the branch-and-bound process. When the branch-and-bound does not find a solution within the currently imposed upper bound and the search is not truncated before its backtrack limit (see infra), the lower bound is set equal to the upper bound plus one in order to prevent that this infeasible upper bound will be used again in the search process. Moreover, it lets us conclude that the current solution is the optimal solution and we can stop the search. Since the branch-and-bound checks all possibilities within that UB and was not truncated, we know that no solution exists with a makespan equal to the upper bound, so the lower bound must be equal to UB+1.

Upper bound As already mentioned above, the branch-and-bound also relies on a (virtual) upper bound (i.e., different from the real upper bound or the best found solution) that is imposed on the makespan value. Each time a job symbol is examined, the branch-and-bound checks if the current completion time of the machine plus the processing time of the job on that machine is not greater than the upper bound value. If so, there is no advantage of scheduling the job on that machine as no better solution will be found (*Step 2*). The initial upper bound (UB) is set equal to the solution of the shortest processing time (SPT) heuristic. During the scheduling process, i.e., each time the evaluation function of the meta-heuristic is called, the upper bound is updated according to a certain strategy. A first strategy is to set the upper bound equal to the best-found solution plus a certain value X. If we add a positive value to the current best solution, we allow the solution to deteriorate by X units. If we add a negative value to this best-found solution, we force the branch-and-bound procedure to find a better solution than the current best one. A second strategy is to fix the upper bound to a random value between the lower bound and the best-found solution. Both strategies will be tested in our computational experiments allowing a wide range of X-values for the first strategy.

Minimal and available processing time Next to this upper bound, the branch-and-bound also examines the solution by making use of two new definitions: the minimal and available processing time. The *minimal processing time* (MPT) is the minimal time we need to schedule all the jobs and is initially equal to the sum of all the shortest processing times of the jobs. The *available processing time* (APT) is the time we have on each machine to schedule the jobs without violating the upper bound and is initially set equal to the upper bound times the number of machines. At each level, the branch-and-bound procedure inspects the difference between the available and the minimal processing time and compares it with the difference between the processing time of the job on the selected machine and its shortest processing time. If the difference between the available and minimal processing time is equal to zero, it means that the remaining jobs should be scheduled on their shortest processing time machine in order not to violate the upper bound. If the difference is positive, there is still more flexibility in scheduling the jobs. The introduction of these new definitions helps the branch-and-bound in examining the solutions more efficiently. This conditional expression is shown in Step 2 of the flow chart of Figure 2 and is equal to

$$LB_2 \leq UB \text{ or } p_{ij} - min_i p_{ij} \leq \text{APT} - \text{MPT}. \tag{1}$$

The available and minimum processing times are updated during the scheduling process by subtracting the corresponding processing and shortest processing time of the scheduled job, respectively (*Step 3*).

Example We can illustrate the implementation details of the branch-and-bound procedure using the example described in Section 2.1 and displayed in Table 1. The initial upper bound of this example is equal to 9, i.e., the maximal machine completion time resulting from scheduling all jobs to the machine on which they have the shortest processing time (= max{3+2, 3+5, 5+4} = max{5,8,9}). The initial minimal processing time (MPT) is equal to 22 (= 3+5+5+3+2+4), the initial available processing time APT is equal to 27 (= 3×9) and the machine completion times are set to zero.

Assume the initial start solution can be represented by 4 5 * 1 3 * 2 6 with a total makespan of 9, then the branch-and-bound is used to search for an improvement of this current solution, and hence, the upper bound is set to 8. In that case the APT is reduced to 24 (= 3×8). The first job selection is job j = 4 since it is the first job in the job list. The branching scheme is based on the machine assignments and starts with the machine where job j is assigned in the initial solution. In our example, job 4 is assigned to machine 1, and the rest of the machines are branched by decreasing order in the processing time of that job. In the second node, the C_i, APT and MPT values are updated (done in step 3 in Figure 2), and the next job that will be selected is the second job $j = 5$ in the initial solution. Again, the machine order for branching is equal to the current machine assignment of job 5, which is machine 1, followed by the other machines by decreasing processing time.

The search continues up to the last node displayed at the lowest level of the tree where job 6 assigned to machine 3. However, this assignment violates the first condition in step 2 of Figure 3 (displayed in the tree as $LB_1 > UB$). However, all other assignments of job 6 to the two other machines also violate the conditions of step 2 and therefore a backtrack is made to the previous level of the tree. At this level, the same condition of step 2 is checked and violated, leading to a backtrack to level 3 of the tree, while at level 2, backtracking is done because of the violation of the second condition of step 2 (displayed in the tree as $LB_2 \geq UB$). This process of branching and backtracking continues until the search returns at level 0. In our example, none of the nodes have found a feasible solution, and hence, the procedure stops with the STOP- condition, meaning that no feasible solution could be found with a objective value of 8 or lower. Consequently, the lower bound can be updated to 9, and since we already have found a solution with a value of 9 (see Table 1), the search can be stopped and optimality is proven. The branch-and-bound tree is shown in Figure 3.

4 Computational experiments

In this section, the computational experiments are described. In Section 4.1, we give more details on the benchmark instances used in our experiments. Section 4.2 elaborates on the implementation of the stand-alone (meta-)heuristics, while Section 4.3 focuses on the hybridization with the truncated branch-and-bound. In Section 4.4, we benchmark our results for the UPMS with the best-known results from the literature. All algorithms were coded in C++ and run on Intel Core$^{\text{TM}}$ i7-3610QM CPU with 2,30GHz RAM.

4.1 Benchmark instances

The data instances for the $R||C_{max}$-problem were generated by Fanjul-Peyro and Ruiz (2010), who developed a comprehensive new data set based on the generation methods

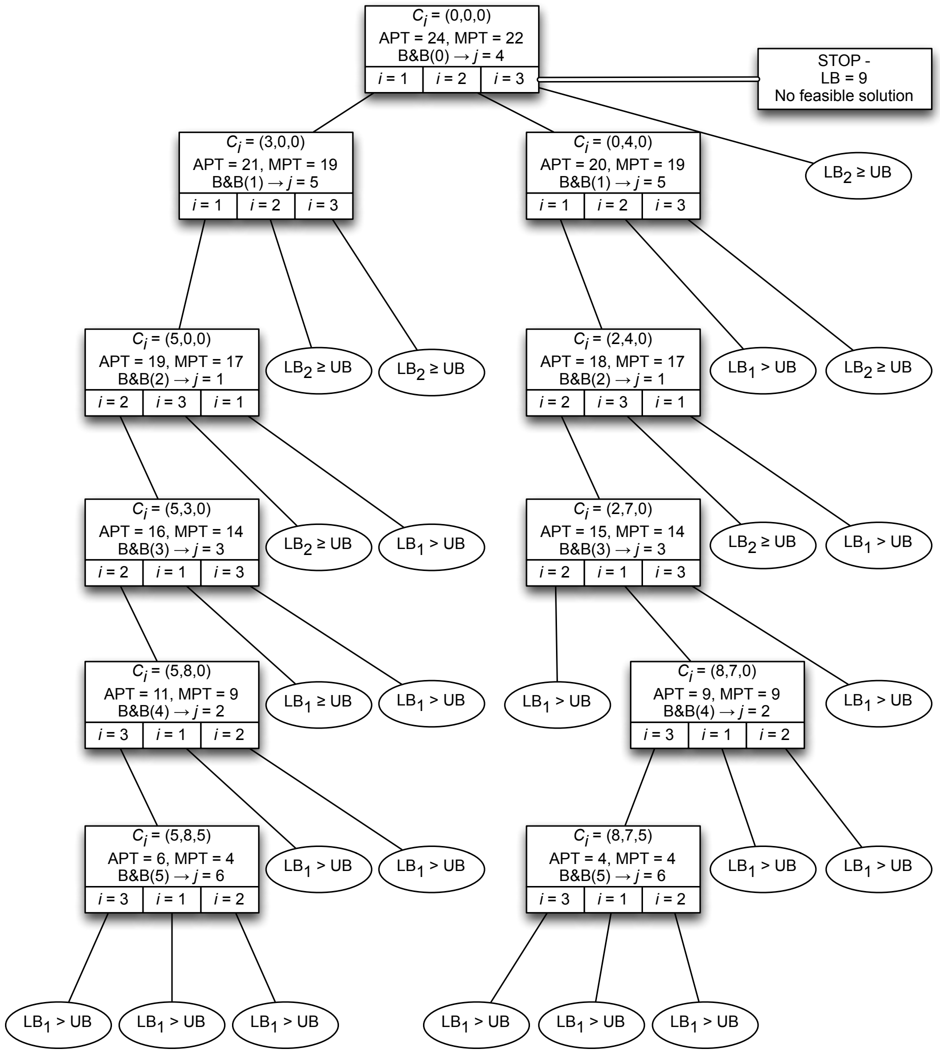


Figure 3: Branch-and-bound tree of the example

of previous work on the UPMS. The authors consider five different uniform distributions for the processing times, i.e $p_{ij} \in$ U(1,100), U(10,100), U(100,200), U(100,120) and U(1,000,1,100). In addition, they consider job correlated (Job Corr.) and machine correlated (Mach. Corr.) processing times. In the job correlated instances, the processing times are determined by the equation $p_{ij} = b_j + d_{ij}$, with b_j and d_{ij} uniformly distributed values in the ranges U(1,100) and U(1,20), respectively. In the machine correlated instances, the processing times are equal to $p_{ij} = a_i + c_{ij}$, with a_i and c_{ij} uniformly distributed in the ranges U(1,100) and U(1,20), respectively (Fanjul-Peyro and Ruiz, 2010). For each of these 7 intervals they consider 5 values of m = 10, 20, 30, 40, 50 and 4 values for n = 100, 200, 500, 1,000. For each combination of m and n, 10 instances were generated. Therefore, in total 200 $\times$ 7 = 1,400 instances were generated. Furthermore, the authors propose benchmark results for these 1,400 instances by solving them with a recent version of ILOG CPLEX (version 11.0). The solver was allowed to run for 2 hours for each individual instance. About 34% of the instances could be solved to optimality with an average gap of approximately 1% (Fanjul-Peyro and Ruiz, 2011). The performances of our algorithms are verified by comparing them with the solutions found by the CPLEX solver. The relative performance (%) can be defined as

$$RP = \frac{C^H_{max}(I) - C^*_{max}(I)}{C^*_{max}(I)}, \tag{2}$$

where $C^H_{max}(I)$ is the makespan value obtained by the heuristic H for instance I and $C^*_{max}(I)$ is the corresponding CPLEX solution for that instance.

All the instances are available at the website *http://soa.iti.es*. Some of the experiments are mainly carried out to determine the best settings of the various parameters of the algorithms, and are therefore tested on a small portion of the dataset. Using 8 randomly selected instances of each dataset (resulting in 56 instances in total) as a training set is better than using the full set to avoid the problem of overtuning our parameters settings. In all our experiments, it is assumed that the tests have been carried out on the full dataset, except where indicated otherwise.

4.2 Performance of stand-alone heuristics

In this section we will analyze the performances of the local improvement techniques and the stand-alone genetic algorithm, tabu search and sampling heuristic, before the hybridization with the branch-and-bound is considered.

4.2.1 Local improvement technique

The four different local search approaches that are described in Section 2.5 were tested to find the best alternative. In order to test their individual performance, all approaches were applied to the solution obtained by the SPT-rule. The results are shown in Table 2. The different columns represent the different local search approaches, with the first column equal to the SPT-solution. The first row gives the average deviations from the CPLEX solutions of Fanjul-Peyro and Ruiz (2010). These values are obtained by taking the average relative performances obtained by equation (2) over all instances. The second row shows the average CPU time (in milliseconds) that the local searches run until no more improvement can be found. The last row shows the number of times (in percentage) that the corresponding local search approach outperforms the other approaches. From this table it can be seen that the second local search that selects jobs from the critical machine and selects machines according to the lowest increase in the job's processing time is the best approach to use. Not only does this local search result in the smallest average deviation, it also requires a reduced CPU time. Moreover, it clearly outperforms the approach proposed in the paper of Fanjul-Peyro and Ruiz (2010) ($LS3$). In order to validate these results, we have applied the local searches on random initial solutions, which resulted in the same overall conclusions and confirmed the outperformance of LS2.

Table 2: Comparison of the local improvement techniques

	SPT	**LS1**	**LS2**	**LS3**	**LS4**
Average % deviation	131.9%	7.6%	3.7%	8.2%	5.9%
Average CPU time (in ms)	3.8	7.4	11.1	71.0	90.8
% best LS approach	0%	9%	70%	11%	32%

4.2.2 Parameter settings

In order to determine the best values for the heuristics without the truncated branch-and-bound, a number of experiments have been carried out. A fractional factorial design was done to determine the best parameter values for the genetic algorithm and the tabu search, and some of the results are shown in this section. For the genetic algorithm, a decision concerning the population size, the crossover type and rate, the mutation type and rate and the seleection type should be made, while for the tabu search, the appropriate list size and neighbourhood selections should be specified. The test values and corresponding results are shown in the tables below. The chosen value is shown each time in bold. All test results in this section are obtained by running our experiments on the training set instead of the full dataset.

Tabu search: In order to determine the best performing neighborhoods, a computational experiment was performed by running the TS with the different NHs restrictions discussed in Section 2.2.2. In these experiments, the list sizes were varied between 1 and 200 and showed that a list size of 100 resulted in the best results (see Table 3). The tests for the performance of the neighbourhoods have been carried out by starting from the most restrictive NH to the NH with the lowest restriction, and finally, to the general pairwise swap without restrictions. More precisely, the most restrictive neighbourhood NH5 is called first, and only when no improvements can be found then NH4 is called. When NH4 is not able to find any improvement, the algorithm tries NH3 to find improvements. This approach is continued with calling NH2 and NH1 and finally, in case of no improvements, a final search for improvemens by the general pairwise swap is executed. Given this sequence of NH search, we have carried out our tests under six different settings, as displayed in Table 4. The column PS shows the average percentage deviation of equation (2) when no restriction is used. The column NH showns the results when the first NH is used in collaboration with the pairwise swap in case NH1 finds no improvements. Likewise, the column NH21 relies on NH2 first, followed by NH1 and PS, and all other columns have a similar meaning. The experiments clearly showed that the TS performs best with the most restricted neighborhood (NH5) and hence this approach has been followed in all the remaining experiments.

Table 3: Parameter fine-tuning for TS: Size of list

1	2	5	10	20	50	100	200
1.53%	1.34%	1.27%	1.06%	0.59%	0.47%	**0.45%**	0.53%

Table 4: Parameter fine-tuning for TS: Use of neighbourhoods

PS	NH1	NH21	NH321	NH4321	NH54321
3.11%	2.52%	2.52%	1.06%	0.77%	**0.45%**

Genetic Algorithm: In order to determine the best values for the parameters of the genetic algorithm, a computational experiment on the training set data instances have been performed varying different settings for the population size, the selection type as well as for the crossover and mutation types and rates.

The types used for the selection, mutation and crossover has been previously discussed in Section 2.3.2 and will be briefly summarized here. The Selection type consists of three variants known as tournament selection (Type 1), roulette wheel selection (Type 2) and ranking selection (Type3) The mutation type also consists of three variants known as swap (Type 1), insertion (Type 2) and inversion mutation (Type 3). The crossover type have five different types consisting of paritally mapped crossover (Type 1), order crossover (Type 2) position based crossover (Type 3), order based crossover (Type 4) and cycle crossover (Type 5). The selection of the best performing types is based on the percentage of times each type leads to the best solution, as is shown in bold in Table 5.

The rates of the best performing mutation and crossover operators have been varied between 0% and 100% as shown in Table 6 and show the best performing values measured by the average percentage deviation of equation (2).

Finally, the best value for the population size has been found by varying the size between a very low value (4) to a very high value (500) and the best results were found with a population size of 8. Our tests have shown an impact of varying the population size according to the number of jobs and machines, but this impact was not significant. Therefore, we have decided to put the population size to the fixed value of 8. Partial results showing the average percentage deviation are shown in Table 7.

Table 5: Parameter fine-tuning for GA: Type of selection (3), mutation (3) and crossover (5)

	Type 1	Type 2	Type 3	Type 4	Type 5
Selection	**71%**	63%	61%	-	-
Mutation	**73%**	57%	55%	-	-
Crossover	54%	**77%**	21%	21%	43%

Table 6: Parameter fine-tuning for GA: Mutation and crossover rate

	0%	10%	25%	50%	75%	90%	100%
Mutation	1.11%	1.27%	1.06%	**1.01%**	1.17%	1.07%	1.02%
Crossover	1.37%	1.09%	1.10%	1.08%	1.08%	**1.06%**	1.10%

Table 7: Parameter fine-tuning for GA: Population size

	4	6	8	10	20	50	100	200	500
Popsize	1.11%	1.25%	**1.01%**	1.12%	1.21%	1.29%	1.23%	1.27%	1.32%

4.2.3 Comparison of heuristics

With the parameter settings and the implementation of the best local search approach discussed in the previous section, the heuristics were run 10 times on all 1,400 instances described in Fanjul-Peyro and Ruiz (2010) for 15 seconds. The results are summarized in Table 8. A distinction is made according to the number of jobs, the number of machines and the distribution of the processing times. Again, the values in the table are the average relative performances obtained by equation (2) over all the instances per problem parameter. The last rows shows the overall relative performance of the heuristics (Average), the number of instances with a better solution (#Improv), the number of instances that have been proven to be optimal (#Opt) and the number of times the procedure shows the best result (%Best). From these rows, it can be seen that the tabu search algorithm (TS) outperforms the sampling (Sam) and the genetic algorithm (GA) procedure.

If we look at the influence of these problem parameters, we see that for all instances, the best performance is obtained for larger problem sizes ($n = 1,000$), a small number of machines ($m = 10$) and large processing time ranges ($p_{ij} \in U(1,000,1,100)$) or job correlated processing time. For these last two categories, the TS is even able to improve the CPLEX solutions of Fanjul-Peyro and Ruiz (2010). Thanks to the good performance of the tabu search algorithm, we have chosen to use this method for hybridization with the branch-and-bound procedure of Section 3, which is the topic of the next section.

4.3 Hybrid procedure parameter setting

In this section, we examine the hybridization of the tabu search heuristic with the truncated branch-and-bound. As mentioned in Section 3.2, there are two important parameters that determine the functioning of the hybrid procedure: the chosen upper bound strategy and the maximum number of backtrack limits. Besides the random strategy, where the upper bound is set equal to a random value between the current best solution and the lower bound, we have examined a large number of possible X-values for the second strategy, with X ranging from -100 to +5. Negative X-values will force the branch-and-bound to find a better solution than the one obtained by the (meta-)heuristic, while a positive value

Table 8: Relative performances of the heuristics

		Sam	**TS**	**GA**
n	100	2.20%	0.38%	1.30%
	200	2.15%	0.71%	1.22%
	500	1.30%	0.53%	0.80%
	1,000	0.77%	0.34%	0.54%
m	10	0.84%	0.11%	0.33%
	20	1.43%	0.37%	0.76%
	30	1.43%	0.46%	0.91%
	40	1.97%	0.78%	1.33%
	50	2.36%	0.73%	1.50%
p_{ij}	U(1,100)	2.59%	1.95%	2.17%
	U(10,100)	2.44%	1.57%	1.71%
	U(100,120)	0.42%	0.00%	0.16%
	U(100,200)	1.72%	0.08%	0.61%
	U(1,000,1,100)	0.19%	-0.01%	0.05%
	Job Corr.	1.11%	-0.53%	0.25%
	Mach. Corr.	2.76%	0.38%	1.81%
	Average	1.61%	0.49%	0.97%
	#Improv	42	249	98
	#Opt	0	9	0
	%Best	10%	96%	27%

allows a deterioration of the makespan value. However, these deteriorations are kept small by limiting X to 5. With respect to the truncation of the branch-and-bound, we let the backtrack limit range from 0 to 1,000,000. Of course, given that we use a time limit as stopping criterion, the choice of the backtrack limit will have an influence on the relative computation times of the branch-and-bound compared to the meta-heuristics. The higher the backtrack limit, the more CPU time will be spent by the branch-and-bound, resulting in less schedules examined by the meta-heuristics. A low backtrack limit on the contrary, will result in a higher percentage of the CPU time that will be used by the meta-heuristics. The branch-and-bound procedure can be called in two different ways in our hybrid TS approach, as follows:

- Every time the algorithm finds a solution improvement, it updates the upper bound and the branch-and-bound procedure is called. Since this only happens rarely during a search, the backtrack limit is set to high values (up to 1,000,000).
- Every time a local search has been carried out, the branch-and-bound is called. Since these call are done very frequently, the backtrack limit has been set to very low values (ranging from 0 to maximum 1,000).

In order to determine the best upper bound strategy and backtrack limit for the hybrid TS/B&B procedure, all test results in this section are obtained by running our experiments on the training set instead of the full dataset under a stopping criterion of 15 and 60 seconds. Only the most relevant results are summarized in Table 9 showing the average deviations from the CPLEX solutions are given for the hybrid tabu search algorithm (HTS).

Table 9: Influence of upper bound strategy and backtrack limit on the performance of the hybrid tabu search (HTS) (partial results)

B&B backtrack limit after UB update	X-value for LS	B&B backtrack limit after LS (15 s.)		B&B backtrack limit after LS (60 s.)	
		no B&B	10	no B&B	10
no B&B	Ran	0.455%	0.462%	0.432%	0.433%
	-1		**0.453%**		0.443%
1,000	Ran	0.533%	0.525%	0.496%	0.374%
	-1		0.464%		0.444%
1,000,000	Ran	0.530%	0.522%	0.497%	**0.371%**
	-1		0.461%		0.441%

The computational results have shown that a backtrack limit of 10 for the B&B procedure performed after each local search (LS) of 10 performs best, and therefore, higher values are not shown in table 9. The backtrack limit for the B&B performed every time a new upper bound have been found is set at 0 (i.e., no B&B, 1,0000 or 1,000,000) and the best value depends on the stop criterion. However, the table also shows that increasing the time limit from 15 to 60 seconds results in improvements, but the improvements are bigger for the tests where the B&B after the UB update is set a high values for the backtrack limit. Finally, the best results are obtained by using the B&B for after each LS and each UB update, leading to an average percentage deviation of 0.371%.

4.4 Computational results of the $R||C_{max}$-problem

The hybrid heuristics with their best X-values and backtrack limit parameter settings were run 10 times on all 1,400 instances of Fanjul-Peyro and Ruiz (2010) for 15 seconds. Table 10 summarizes the results. The pure meta-heuristic results are the results for the procedures used in Table 8, while the hybrid meta-heuristics rely on the branch-and-bound procedure of Section 3 added on top of the pure meta-heuristic searches. This table confirms the improvement of the stand-alone heuristics by the B&B hybridization and the dominance of the tabu search method.

Table 10: Relative performances of the heuristics

	Sam	**TS**	**GA**
Pure meta-heuristic	1.61%	0.49%	0.97%
Hybrid meta-heuristic	**1.52%**	**0.47%**	**0.91%**

The best approach for each heuristic was compared with the CPLEX-solutions of Fanjul-Peyro and Ruiz (2010) (see Table 11). Again, we made a distinction between the number of jobs, number of machines and the distribution of the processing times. Our tests revealed that the TS was able to obtain better solutions than the CPLEX-solutions in 245 instances (see Table 11). These instances mainly had job correlated processing times or processing times from the uniform distribution U(1000,1100), number of jobs larger than 200 and number of machines larger than 30. This validates the benefit of a meta-heuristic for larger scaled problems. The poor performance of CPLEX for these instances (especially the job correlated instances), was also conformed by the authors (Fanjul-Peyro and Ruiz, 2010).

Table 11: Number of times the procedures find better solutions than the CPLEX solutions of Fanjul-Peyro and Ruiz (2011)

		HSam	**HTS**	**HGA**
n	100	0	48	3
	200	9	54	18
	500	9	71	38
	1,000	25	72	40
m	10	0	5	0
	20	1	23	11
	30	15	90	33
	40	13	72	32
	50	14	55	23
p_{ij}	U(1,100)	0	0	0
	U(10,100)	0	0	0
	U(100,120)	2	30	3
	U(100,200)	5	23	7
	U(1,000,1,100)	6	61	13
	Job Corr.	30	129	76
	Mach. Corr.	0	2	0
Total		**43**	**245**	**99**

These best heuristics were also compared with the best-known solutions from the literature. These are the solutions obtained by the partial enumeration of Mokotoff and Jimeno (2002) (*Partial*), the recovering beam search of Ghirardi and Potts (2005) (*RBS*) and the iterated greedy local search based heuristic of Fanjul-Peyro and Ruiz (2010) (*NVST-IG+*). Since the results presented in the paper of Fanjul-Peyro and Ruiz (2011) were obtained by parallel algorithms ran on a cluster of dual processors, we did not include them in our overview to have a fair comparison. However, we are aware of these improved results and we see the use of parallel algorithms as a noteworthy future research topic.

The results presented in the left part of Table 12 were taken from the paper of Fanjul-Peyro and Ruiz (2011). The authors ran the algorithms of Mokotoff and Jimeno (2002) and Ghirardi and Potts (2005) on their benchmark set using a stopping criterion of 15 sec. However, both the partial enumeration as the recovering beam search, on average, needed more CPU time. The values in the right part of the table are our proposed algorithms. All values in the table are presented as the average deviations from the CPLEX solutions per category of processing time distributions. The best values are indicated in bold, the second best values are underlined.

These values show that we are able to compete with, and in some cases to outperform, the best known results from the literature as given by the NVST-IG+ procedure of Fanjul-Peyro and Ruiz (2010). Although our heuristics are not always able to outperform the NVST-IG+ procedure on all processing time distributions, they all obtain better results for the job correlated and machine correlated processing time instances, as well as for the U(100,120), U(100,200) and U(1000,1100)-instances. Furthermore, we approximate the performances on the U(1,100) and U(10,100) instances with TS procedure. In addition, in average, our heuristics have better results compared to either the more time consuming partial enumeration procedure of Mokotoff and Jimeno (2002) or the recovering beam search of Ghirardi and Potts (2005).

5 Conclusions

This paper considered the problem of scheduling a number of jobs on a number of unrelated parallel machines in order to minimize the makespan. We have developed a genetic algorithm and a tabu search algorithm, which were hybridized with a truncated branch-and-bound procedure in order to accelerate the search process to near-optimal solutions. Both meta-heuristics were further improved by means of an effective local search algo-

Table 12: Comparison with benchmark results from the literature

	Partial	**RBS**	**NVST-IG+**	**HSam**	**HTS**	**HGA**
U(1,100)	2.88%	2.03%	**1.34%**	2.51%	1.83%	2.02%
U(10,100)	1.31%	1.87%	**0.75%**	2.31%	1.51%	1.64%
U(100,120)	0.33%	0.13%	0.04%	0.36%	**0.00%**	0.14%
U(100,200)	1.05%	0.81%	0.32%	1.52%	**0.08%**	0.56%
U(1000,1100)	0.23%	0.18%	0.02%	0.17%	**-0.01%**	0.04%
Job Corr.	2.34%	0.35%	0.48%	1.06%	**-0.53%**	0.23%
Mach. Corr.	0.94%	2.36%	0.55%	2.71%	**0.38%**	1.77%
Average	1.3%	1.10%	0.50%	1.52%	**0.47%**	0.91%

rithm and were combined in order to take advantage of both heuristics. The truncated branch-and-bound procedure checked the possibility of scheduling the solutions obtained by the meta-heuristics within an upper bound set on the makespan value. Each individual job-machine assignment was successively examined by the branch-and-bound and altered if necessary. In doing so, the meta-heuristics were stimulated to skip non-promising areas of the solution space. We compared the performances of these heuristics on a standard data set available in the literature. We carefully examined the influence of the different heuristic parameters, such as the backtrack limit and the upper bound strategy, to select the best combination for the hybrid algorithms. These experiments showed that each heuristic performed best with a different upper bound strategy and backtrack limit.

Our computational experiments revealed that the tabu search heuristic outperformed the genetic algorithm, but that hybridization with the truncated branch-and-bound procedure was able to compete with and sometimes outperform the best known results from the literature. For the job correlated and machine correlated processing time instances as well as for the bigger instances, our heuristics were even able to find better solutions than the CPLEX solutions.

An interesting field of future research could be to develop an exact branch-and-bound procedure, relying on the knowledge obtained by the truncated branch-and-bound procedure, and to benchmark this procedure against our hybrid procedures and the methods described in the literature. Moreover, the extension of our procedures to more complex problems, such as the inclusion of setup times is definitely an interesting future research topic.

6 Acknowledgements

We acknowledge the support given by the “Bijzonder Onderzoeks Fonds (BOF), Vlaanderen, Belgium” under contract number BOF07/24j/034.

References

Cheng, R. and Gen, M. (1997). Parallel machine scheduling problems using memetic algorithms. *Computers & Industrial Engineering*, 33:761–764.

Fanjul-Peyro, L. and Ruiz, R. (2010). Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, 207:55–69.

Fanjul-Peyro, L. and Ruiz, R. (2011). Size-reduction heuristics for the unrelated parallel machines scheduling problem. *Computers & Operations Research*, 38:301–309.

Garey, M. and Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman & Co., San Francisco.

Gendreau, M. and Potvin, J.-Y. (2010). *Handbook of Metaheuristics.* Springer.

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

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

Glover, F. and Laguna, M. (1997). *Tabu Search.* Kluwer Academic Publishers.

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

Guo, Y., Lim, A., Rodrigues, B., and Yang, L. (2007). Minimizing the makespan for unrelated parallel machines. *International Journal on Artificial Intelligence Tools*, 16:399–416.

Holland, J. (1975). Adaptation in natural and artificial systems. *University of Michigan Press, Ann Arbor.*

Kellegöz, T., Toklu, B., and Wilson, J. (2008). Comparing efficiencies of genetic crossover operators for one machine total weighted tardiness problem. *Appied Mathematics and Computation*, 199:590–598.

Laguna, M., Barnes, J., and Glover, F. (1991). Tabu search methods for a single machine scheduling problem. *Journal of Intelligent Manufacturing*, 2(2):63–73.

Lin, Y., Pfund, M., and Fowler, J. (2011). Heuristics for minimizing regular performance measures in unrelated parallel machine scheduling problems. *Computers & Operations Research*, 38:901–916.

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

Mendes, A., Müller, F., França, P., and Moscato, P. (2002). Comparing meta-heuristic approaches for parallel machine scheduling problems. *Production Planning & Control*, 13:143–154.

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

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

Monien, B. and Woclaw, A. (2006). Scheduling unrelated parallel machines computational results. *Lecture Notes in Computer Science*, 4007:195–206.

Piersma, N. and Van Dijk, W. (1996). A local search heuristic for unrelated parallel machine scheduling with efficient neighborhood search. *Mathematical and Computer Modelling*, 24:11–19.

Reeves, C. (2003). *Handbook of Metaheuristics*, chapter Genetic algorithms, pages 55–82. Kluwer Academic Publishers, Dordrecht.

Shin, H.-J., Kim, C.-O., and Kim, S.-S. (2002). A tabu search algorithm for single machine scheduling with release times, due dates, and sequence-dependent set-up times. *International Journal of Advanced Manufacturing Technology*, 19:859–866.

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

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

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