Setting a common due date in a constrained flowshop: A Variable Neighbourhood Search approach [*]

Paz Perez-Gonzalez[†] Jose M. Framinan

Industrial Management, School of Engineering,

University of Seville. Camino de los descubrimientos s/n. 41092 Seville, Spain

Abstract

In this paper we study a due date setting problem in a flowshop layout. The problem consists of scheduling a set of jobs arriving to the system together with jobs already present (denoted as old jobs), in order to set a common due date for the new jobs. Since the old jobs have a common due date that must not be violated, our problem is a rescheduling problem with the objective of minimising the makespan of the new jobs (thus obtaining the tightest possible due date for the new jobs) and a constraint since the maximum tardiness of the old jobs must be equal to zero. This approach leads to an interesting scheduling problem in which two different objectives are considered, each one for a subset of the jobs that must be scheduled. To the best of our knowledge, this type of problems have been scarcely considered in the literature, and only for very specific purposes. Since our problem is clearly NP-hard, a new heuristic based on Variable Neighbourhood Search (VNS) has been designed. The computational results show that our proposed heuristic outperforms two existing heuristic methods for similar problems in the literature.

[*]This is an Accepted Manuscript of an article published by Elsevier in Computers and Operations Research Volume 37, Issue 10, October 2010, Pages 1740-1748, available online: http://dx.doi.org/10.1016/j.cor.2010.01.002

[†]Corresponding author. Tel. +3495 448 7327; fax: +3495 448 7329. E-mail address: pazperez@esi.us.es

Keywords: Scheduling; Metaheuristics; Flowshop; Common due date; Due date setting.

1 Introduction

Manufacturing environments are complex and dynamic, implying to respond effectively to satisfy customers orders, which means providing the customers with tight and reliable due dates (Framinan, 2008). For this reason, the due date setting problem is an important issue in the order capture activities together with order acceptance/rejection and order scheduling (Framinan and Leisten, 2009).

This paper considers a dynamic scenario, where machines arranged in a flowshop are busy by processing a previously scheduled set of jobs belonging to an already committed order with a given common due date. The problem is to set a common due date for a new incoming order (consisting of a new set of tasks or jobs) by adequately scheduling these new jobs. Note that the overall goal of the company is to meet the proposed due dates, which depend not only on the scheduling procedure followed, but also on the 'reasonableness' of the due dates (Ragatz and Mabert, 1984), defined by Vig and Dooley (1991) as a measure of the due date performance reflected on the capability of the system to achieve successfully an arbitrary set of due dates. There are two aspects of due date performance Framinan (2009): delivery reliability which is the ability to consistently meet promised due dates (Cheng and Jiang, 1998), given the critical importance of the fulfilment of the promised due dates (Framinan, 2007); and delivery speed, which is the ability to deliver orders to the customers with shortest lead times (Philipoom, 2000). In order to verify both aspects, we minimise the maximal completion time or makespan of jobs of the new order, guaranteeing the delivery speed, and imposing that the due date of the old jobs is not violated, achieving delivery reliability. Two different scenarios can be considered:

- The jobs already in the system are regarded as 'frozen' (i.e., their schedule cannot be modified). The resulting problem is then to schedule the new set of jobs in order to obtain a tight due date (e.g., by minimising their makespan) taking into account that the machines are not immediately available (see Perez-Gonzalez and Framinan, 2009), or

- to allow modifying the schedule of the existing jobs in the system and to reschedule them together with the new jobs in order to obtain a tight due date for the latter set, as long as the due date already committed for the existing jobs is not violated.

In the latter case, the resulting problem can be considered a type of rescheduling problem. Clearly, this option may potentially result in tighter due dates than the previous ones, and implies a constrained problem since the common due date of the old jobs is a deadline. The allowance of rescheduling committed orders is one of the mechanism described by Framinan (2008) to provide both tight due dates and a detailed, reliable production schedule for the shop floor, which are important aspects for the company as it has been previously explained.

The rest of the paper is organised as follows: Section 2 describes the problem and introduces the related literature and those problems identified in the literature as similar to our problem. Section 3 shows the different solution procedures that can be applied to solve the problem, including a new proposal based on VNS. These procedures are calibrated in Section 4 and compared by carrying out the computational experience presented in Section 5. Finally, conclusions and future research lines are summarised in Section 6.

2 Problem statement and related problems

The problem considered is a due date setting problem in which the scheduler may update the existing schedule upon the arrival of new jobs into the system. As denoted in the related literature, jobs are categorised either as 'old' or 'new' jobs. The set of n_O old jobs, J_O, belongs to a previously scheduled order. Therefore, they have a common due date d which has already been set and it is considered to be a parameter in our problem. A set of n_N new jobs, J_N, arrives to the system. Our objective is to set a common due date for J_N by scheduling all n jobs, $J = J_O \bigcup J_N$ with $n = n_O + n_N$.

In order to set a tight common due date for the new jobs, the objective considered is to minimise the maximum completion time (makespan) for J_N, denoted as $C_{max}^{J_N}(S)$, with S a sequence composed by jobs belonging to J. Furthermore, the existing jobs have a common due date d which cannot be violated. Therefore, a feasible sequence S is a sequence in which the completion times of jobs in J_O are less or equal than their common due date d. This is equivalent to impose that the maximum tardiness of S for jobs in J_O is zero, i.e., $T^{J_O}_{max}(S) = \max T^{J_O}_j(S) = 0$ being $T^{J_O}_j(S) = \max\{0, C^{J_O}_j(S) - d\}$ the tardiness of the job $j \in J_O$, and $C^{J_O}_j(S)$ the completion time of the job $j \in J_O$ in the last machine regarding the sequence S. This is also equivalent to $T^{J_O}(S) = \sum_{j \in J_O} T^{J_O}_j(S) = 0$, i.e., the total tardiness of S for jobs in J_O is zero.

Our work focuses on the flowshop, implying a natural ordering of the m machines in the shop in such a way that the jobs go through the same machines in the same order. In general, there are $(n!)^m$ schedules to be considered. However, there is a simplified version of the problem applicable to many situations in which it is assumed that the processing sequence of the jobs is the same for all machines (i.e., permutation flowshop) and hence only $n!$ schedules have to be considered.

Using the notation defined by Graham et al. (1979), our constrained rescheduling problem, called CRP, can be denoted as $Fm|prmu, d_j = d|C^{J_N}_{max}/T^{J_O}_{max} = 0$, where Fm indicates a flowshop with m machines, $prmu$ denotes the permutation case, $d_j = d$ specifies the use of a common due date, and finally, $C^{J_N}_{max}/T^{J_O}_{max} = 0$ is the constrained objective function. It is easy to see that this problem is strongly NP-hard for more than two machines, as if we assume that $J_O = \varnothing$, then it may be reduced to the classical permutation flowshop scheduling problem with makespan objective, denoted as CP, which is known to be strongly NP-hard (Garey et al., 1976).

To the best of our knowledge, CRP has not been studied in the scheduling literature, as we consider a different objective for each set of jobs. However, some related problems can be identified. On one hand, CP is the first problem similar to CRP, since CP is a special case of CRP if $J = J_N$. In order to determine the difficulty degree of our problem CRP, Perez-Gonzalez (2009) compares this problem with CP by analysing the structures of solutions of both problems, solving exactly small instances, and concluding that CRP is statistically more difficult than CP for realistic values of the common due date d (around a 40% from the optimal makespan of the jobs belonging to J_O), and for those cases where the size of both sets are similar or equal. Therefore, in order to obtain good solutions for real sized instances, approximate procedures have to be considered.

On the other hand, in the rescheduling literature, some references consider similar problems, most of them for a single machine. Unal et al. (1997) address the single machine rescheduling problem with part-type dependent setup times, where new jobs are inserted in the sequence of the old jobs without violating the due date of these old jobs. Hall and Potts (2004) and Mocquillon et al. (2008) study a problem similar to the previous one, but they measure the disruption of the schedule by means of a cost function. Yang (2007) schedules both new and old jobs considering a disruption cost. Rangsaritratsamee et al. (2004) reschedule the set of jobs available in a dynamic job shop at each rescheduling point minimising their makespan. Since these approaches focus on maintaining stability of the sequence of the old jobs, these problems are not very similar to CRP and the corresponding solution methods cannot be adapted.

There are also similar approaches in the multi-criteria scheduling literature. In Minella et al. (2008) this kind of problems are reviewed for the flowshop layout, including some problems with due date related objectives, although they assume that the two objectives apply both to all jobs in J. One of these problems is the permutation flowshop scheduling problem with the objective of minimising the makespan subject to a given maximum tardiness. This problem is denoted as $Fm|prmu|\varepsilon(C_{max}/T_{max})$ (following the notation of T'kindt and Billaut, 2006), indicating that the objective is to minimise the makespan while obtaining a maximum tardiness less or equal than a given value ε. For convenience, we denote this constrained scheduling problem as CSP. CSP may be seen as a special case of a more general problem denoted $Fm|prmu|\varepsilon(Z/T_{max})$ in which $Z = \lambda C_{max} + (1-\lambda)T_{max}$, $\lambda \in [0, 1]$. We denote this problem as Generalised CSP or $GCSP$, being CSP the case when $\lambda = 1$. Differences among CRP, CP, CSP and $GCSP$, according to the set of jobs involved in the problem, the due date, and the objectives considered are summarised in Table 1.

Given the similarity of these problems, the methods applied to CP, $GCSP$ or CSP could be adapted to our problem CRP. We intend to provide a heuristic method to solve the problem. In order to determine the efficiency of our proposal, we adapt the best

Problem	Notation	Set of jobs	Due dates	Objective
CRP	$Fm\|prmu, d_j = d\|C_{max}^{J_N}/T_{max}^{J_O} = 0$	$J = J_O \bigcup J_N$	Common	$C_{max}^{J_N}$ subject to $T_{max}^{J_O} = 0$
CP	$Fm\|prmu\|C_{max}$	J	No	C_{max}^{J}
CSP	$Fm\|prmu\|\epsilon(C_{max}, T_{max})$	J	Different	C_{max}^{J} subject to $T_{max}^{J} \leq \epsilon$
$GCSP$	$Fm\|prmu\|\epsilon(Z, T_{max})$	J	Different	$\lambda C_{max}^{J} + (1-\lambda)T_{max}^{J}$ subject to $T_{max}^{J} \leq \epsilon$

Table 1: Differences between CRP, CP, CSP and $GCSP$.

solution procedure of each related problem in the following subsections.

2.1 Solution procedures for CP: Adaptation for CRP

A great number of exact and approximate methods have been applied to solve CP. Regarding approximate methods, a comparative evaluation of heuristics and metaheuristics is presented by Ruiz and Maroto (2005), and a classification of the heuristics is carried out by Framinan et al. (2004).

The Iterated Greedy (IG) algorithm proposed by Ruiz and Stützle (2007) has proved to be among the most effective methods to solve CP. IG is a rather new heuristic method which generates a sequence of solutions by iterating over greedy constructive heuristics using two main phases: destruction and construction. IG is closely related to the Iterated Local Search procedure presented by Lourenco et al. (2002), and it is easily tunable and applicable to other problems because its simplicity.

Here we simple describe briefly the IG algorithm and for more details we refer to Ruiz and Stützle (2007). The initial sequence is constructed by NEHT, the improved version presented by Taillard (1990) of the NEH heuristic. NEH (Nawaz et al., 1983) is the best constructive heuristic for CP (Turner and Booth, 1987). This well-known heuristic is based on the idea that jobs with longest processing time on all machines should be scheduled as earliest as possible, so an initial sequence is built considering this idea, and then a schedule is constructed iteratively from the initial sequence. The original complexity of the NEH is $O(n^3m)$, being reduced by Taillard (1990) to $O(n^2m)$ by means of calculating the makespan for all partial schedules for a given iteration in a single step. Once the initial sequence is provided by the NEHT, the IG applies the destruction procedure removing δ jobs chosen randomly without repetition, and the construction phase inserts them in the same way that NEHT. Then, the local search procedure, called Iterative Improvement, improves each solution generated in the construction phase. This procedure is based on the insertion method, which is commonly regarded as a very good choice for CP (Taillard, 1990). The last step is to accept or not the new sequence as the incumbent solution for the next iteration. Ruiz and Stützle (2007) consider a simple simulated annealing-like acceptance criterion, similarly to the one presented by Osman and Potts (1989), with a constant temperature which depends on a given parameter T.

The adaptation of this method to CRP implies minor modifications:

- Since the algorithm has been originally developed for an unconstrained problem, a natural adaptation considers only feasible solutions by checking the feasibility of each solution in the construction and local search procedures, thus rejecting unfeasible solutions.

- As a consequence, the objective function to evaluate each solution S in our adaptation is $C_{max}^{J_N}(S)$, i.e., only the makespan of the new jobs is considered.

- For our problem, the initial solution has to be feasible and it is generated by the so-called Initial Feasible Solution method. This procedure uses an adaptation of the NEHT, called ANEHT, to the permutation flowshop problem with initial availability constraint (Perez-Gonzalez and Framinan, 2009), which considers the machine availability instants (a_i) to compute the makespan. The Initial Feasible Solution method generates the machine availability instants a_i for each machine i from 1 to m as the completion time for each machine of the sequence S^{J_O}, given by the NEHT applied to J_O. Then, the method applies ANEHT to J_N and constructs $S = S^{J_O} \bigcup S^{J_N}$, guaranteeing that S is feasible, as the common due date of J_O is obtained from the makespan value given by S^{J_O}.

2.2 Solution procedures for $CSP/GCSP$: Adaptation for CRP

Table 2 shows a summary of the reviewed references tackling both $GCSP$ and CSP. This table indicates each problem and the method proposed to solve it.

Authors	**Problem**	**Resolution Method**
Daniels and Chambers (1990)	CSP	Constructive heuristic (DC)
Chakravarthy and Rajendran (1999)	$GCSP$	Adaptation of DC (CR)
Allahverdi (2004)	$GCSP$	Heuristic APH
Framinan and Leisten (2006)	CSP	Heuristic FL
Ruiz and Allahverdi (2009)	$GCSP$	Heuristic SGAT

Table 2: Resolution methods for problems $GCSP$ and CSP.

The best heuristic known to solve the general problem $GCSP$, and consequently CSP, is the one proposed by Ruiz and Allahverdi (2009), based on a Genetic Algorithm (GA). This heuristic is called Steady State GA, SGAT.

Genetic Algorithms (GAs) have been frequently used in the scheduling research community since the mid 1990s with very good results. GAs mimic the natural selection and evolution of species, abstracting a solution for a specific problem and encoding it into the chromosome of an individual. In general, for applying GAs in scheduling, the individuals are sequences, and a population is a set of sequences. The GA presented by Ruiz and Allahverdi (2009) for the $GCSP$ is based on a new type of steady state GAs, called SGAT (Ruiz et al., 2006). In a steady state GA, there is only one population and new individuals do not replace their parents, but a new individual replaces the worst individual of the population if this new one is unique and better than the worst one. Ruiz and Allahverdi (2009) incorporate significant speed-ups in the local search and a novel three-phase fitness evaluation, specially tailored for dealing with unfeasible solutions.

SGAT is fully detailed in Ruiz and Allahverdi (2009), so we only give an outline here. This algorithm allows unfeasible solutions in the population by defining three states for the population: one state where all solutions are feasible, another state where there are feasible and unfeasible solutions, and another one with all solutions unfeasible. The fitness value of each individual may be calculated depending on each state (see Ruiz and Allahverdi (2009) for details). The initial population is generated constructing two super-individuals, one obtained by the EDD (Earliest Due Date) rule and the other by the NEHT method, and the rest of the population is generated randomly. Once the fitness is computed for all individuals of the population, the parent selection is carried out by a fast and simple selection operator, the n-tournament selection procedure (Ruiz and Allahverdi, 2007), which depends on a pressure parameter. Then, the two parents are crossed by the two-point crossover procedure (TP) (see Michalewicz, 1994 for details), with a probability p_C. A mutation operator consisting of extracting one job from the individual and re-inserting it in another random position is applied to the obtained individual with probability p_M. SGAT also incorporates a Light Local Search scheme which is applied to the best solution in the initial population and also to each offspring generated after the crossover and mutation. Only a fraction p_{LS} of offsprings undergo local search. Finally, each new individual is checked to guarantee its 'uniqueness' once the fitness has been calculated in order to avoid clones in the population.

Given the similarity of both problems, only minor modifications have to be implemented to adapt SGAT to CRP:

- In the first step of the original SGAT, the EDD rule sorts the jobs in increasing order of their due dates, but in our case we have a common due date for all jobs of J_O, therefore it is not possible to apply it. Instead we generate an initial solution by applying the Initial Feasible Solution method previously described for IG. The second super-individual is generated by NEHT applied to $J = J_O \bigcup J_N$.

- The makespan of J_N is used to evaluate the objective function in the first state of the fitness function (all individuals of the population are feasible). For the second state (mixture of unfeasible and feasible solutions in the population) we have used the same method than Ruiz and Allahverdi (2009), adding the objective value of the worst feasible solution to that of the unfeasible solutions, so the best unfeasible solution has worse fitness than the worst feasible solution in the population. Finally, in the third state (all individuals are unfeasible), the objective function is the tardiness of J_O.

- The original Light Local Search method extracts all jobs in a given solution at random and without repetition, one by one, and re-inserts them in all feasible positions of the sequence in order to select the best feasible one. The local search stops when all jobs have been tried in all positions. We apply a modified version of Taillard's improvement to speed up the reinserting method for CRP. Our implementation checks all positions of the sequence when the job to be inserted belongs to J_N, as in the original version (see Breit (2004)). However, it discards unfeasible positions if the job involved in the insertion belongs to J_O, i.e., it checks all positions until that which provides the completion time greater or equal to the due date d.

3 Proposed metaheuristic: Refreshing VNS

In the previous section, we adapted the best existing methods from the literature for those problems identified as similar to our problem. In this section we propose a new heuristic for CRP based on the Variable Neighbourhood Search (VNS), named Refreshing VNS.

VNS is a metaheuristic based on changing the neighbourhood in a local search procedure. It was developed by Mladenovic and Hansen (1997), and, as opposed to other heuristics based on local search methods, VNS does not follow a trajectory, but explores increasingly distant neighbourhoods of the current solution, changing to a new one when no further improvements are found (Hansen and Mladenovic, 2001). Some authors have used this metaheuristic in flowshop scheduling problems, being all references very recent (see e.g. Framinan and Leisten, 2007; Pan et al., 2008 and Blazewicz et al., 2008). In most references, a hybrid version is developed in which VNS is combined with another heuristic such as IG, SA or TS, among others.

In our case, we present a variant called Refreshing VNS, RVNS, with some novel features. In principle, VNS only considers feasible solutions, but our approach is aimed to problems in which there may be unfeasible solutions in the neighbourhoods. Most procedures dealing with problems with unfeasible solutions introduce a penalisation in the objective function. Therefore, the objective function must change depending on whether the solution is feasible, or not. In some preliminary versions of the heuristic, we penalise the objective function in the case of unfeasible sequences by adding the maximum tardiness of J_O to the makespan value of J_N. However, this approach did not provide good results, so we decide to define two sets of sequences:

- Strict sequences. The feasibility of this kind of solution is guaranteed and can be a solution for the constrained problem.
- Relaxed sequences, which can be either feasible or unfeasible. These sequences are considered for the construction of neighbourhood structures. In general, they are not solutions for the constrained problem. Only some relaxed sequences are selected in the heuristic to check their feasibility. In this case, if a relaxed sequence is feasible, it is considered as strict one.

By using this approach, we can use the same objective function both for strict and relaxed sequences, i.e., the makespan of jobs belonging to J_N, $C_{max}(J_N)$, avoiding the distortion of the objective function. This also allows finding good solutions on the neighbourhoods of the relaxed sequences in a faster manner. Note that only strict sequences can be final solutions of our problem, since only for this kind of sequences is guaranteed the feasibility.

As we consider relaxed sequences, which can be either feasible or unfeasible, our variant of VNS tries to repair good unfeasible solutions (similarly to the idea proposed by Allahverdi, 2004), i.e., those relaxed solutions with a good makespan value but proved as unfeasible, in order to find feasible solutions in the neighbourhood of those good solutions. Moreover, we include an escape method, so RVNS becomes a multi-start method (Lee, 1991), as it restarts the first step from a random solution when the algorithm is trapped in a local minimum. Figure 1 shows the pseudo-code of the Refreshing VNS.

In our method, three major steps are identified: i.e.: Initialisation, Shaking and Change or not, and Local Search/Escape. This is the general structure of a Basic VNS (see e.g., Hansen and Mladenovic (2001)), but we add the Escape method. These are discussed in detail in the next subsections:

```
procedure Refreshing VNS
    % STEP 1:  Initial Solution
        $S^{best}$ := Initial_Feasible_Solution;
        $S$ := NEHT($J$);
        if $S$ is feasible & $C^{J_N}_{max}(S) < C^{J_N}_{max}(S^{best})$ then
            $S^{best}$ := $S$;
        end if
    while (stop criterion is not verified) do
    % STEP 2:  Shaking and Change or not
        for $k = 1$ to $k_{max}$ do
            Shaking($k$, $S$, $S^{best}$, $change$)
            if $change = true$ then
                break;
            end if
        end for
    % STEP 3:  Local Search / Escape
        if $change = true$ then
            General_ LS($S$, $S^{best}$);
            if $S$ is feasible then
                Intensify($S$, $S^{best}$);
            else
                Repair($S$, $S^{best}$);
            end if
        else
            if $S$ is feasible then
                Escape($S$, feasible, $S^{best}$);
            else
                Escape($S$, unfeasible, $S^{best}$);
            end if
        end if
    end while
    return $S^{best}$;
end
```

Figure 1: Pseudo-code of Refreshing VNS.

3.1 Initialisation

Our method starts with two sequences. The first is the best sequence, denoted as S^{best}. This sequence must be strict, so it is generated by the Initial Feasible Solution procedure explained previously, guaranteeing its feasibility. In order to diversify the search method, we generate a relaxed solution S, obtained by the NEHT applied to all jobs in J. If S is feasible and better than S^{best}, then $S^{best} := S$.

3.2 Shaking and Change or not

A Shaking method generates a random solution of the $k-th$ neighbourhood of the current solution. Unlike the exploration of the neighbourhood, which finds the best solution of the $k-th$ neighbourhood, implemented in preliminary versions of the heuristic, Shaking provides better and faster results. Our Shaking method is applied for each neighbourhood structure with size k, with $1 \leq k \leq k_{max} \leq n$, being n the jobs in the system. Shaking considers k neighbours of S, and for each k, k jobs are selected at random, removed and inserted on a new random position of S one by one. For each neighbour S', if it is feasible and better than S_{best}, then the latter is updated.

Change or not allows to apply or not a local search in the case that an improvement is achieved. This method sets the boolean flag *change* as *true* when S' is better than S. In this case the Local Search is applied. Otherwise, an Escape procedure is carried out. Both Local Search and Escape procedures are detailed in the next subsection.

3.3 Local Search/Escape

Several local search procedures may be applied to the relaxed sequence after the Shaking in the case that *change* equals *true*. We have applied more than one type of local search because of the nature of the solutions. As we work with relaxed sequences, the idea is to search exhaustively in their neighbourhoods regardless their feasibility, in order to find better relaxed solutions. This local search is named General Local Search. Once it is applied, we check the feasibility of the current solution. If it is feasible, then the Intensify method is applied, trying to find better feasible sequences in the neighbourhood of the current solution. Else, the Repair method tries to modify the current unfeasible solution in order to make it feasible. These methods are detailed below:

- The General Local Search is an iterative improvement method, where each job of the given sequence S is selected at random and inserted by using Taillard's improvement method in all possible positions of S if it belongs to J_N, or in all feasible positions of S if it belongs to J_O. S^{best} is updated if the sequence obtained is feasible and better. This procedure is repeated until no further improvement is found.

- If the relaxed solution obtained by the General Local Search is feasible, then the Intensify method tries to improve it. First, when the makespan of the jobs in J_N for the given sequence S is lower than the common due date of J_O, i.e., all jobs are sequenced before the due date, the Special Case method is applied. This method is another iterative improvement method similar to the General Local Search. In this case, if the job randomly selected from S belongs to J_N, then it is reinserted in the position before the last job belonging to J_N which provides the best sequence S'. If the job randomly selected from S belongs to J_O, then it is reinserted in the position after the last job belonging to J_N which provides the best sequence S'. If the makespan of the jobs in J_N is lower than d for the new sequence S', the method updates S^{best} if S' is feasible and better than S^{best}. Moreover, if the makespan of the jobs in J_N for S' is lower than the makespan of the jobs in J_N for the given sequence S, then $S := S'$ and another iteration is done. Regardless the Special Case is applied or not, the Intensify method selects randomly each job from the given sequence S belonging to J_N and reinserts it in the position before the last job belonging to J_N which provides the best permutation S'. If the makespan of the jobs in J_N for S' is lower than the makespan of the jobs in J_N for the given sequence S, then $S = S'$, and S^{best} is updated if S is feasible and better.

- If the solution obtained by the General Local Search is unfeasible, then the objective is to repair it. This is done by removing the tardy jobs and inserting them in new positions chosen at random. If the new sequence is feasible and better than S_{best}, then it is adopted as the best solution.

If there is no improvement after Shaking, the local search methods cannot be applied because the heuristic is trapped a local minimum. Instead, an Escape procedure allows escaping from this local optima.

Depending on the feasibility of the current solution, a high number of jobs belonging to the corresponding sets J_O or J_N are removed and inserted in new positions. The percentage of jobs to be removed, q, is randomly generated in a given interval of n_O or n_N, depending on the feasibility of the sequence. If it is feasible, then $\lceil q \cdot n_N \rceil$ jobs in J_N are removed and inserted according to the following rule: the first job selected at random is scheduled in the first position, the second in the second position, and so on. If the sequence is unfeasible, then $\lceil q \cdot n_O \rceil$ jobs of J_O are removed and inserted by using the same procedure. For both cases, if the sequence S obtained is feasible and better than S^{best}, then S^{best} is updated with S.

4 Experimental calibration of the algorithms

Since some of the algorithms to be tested for our problem are adaptations of existing algorithms for similar problems, it may be that a different combination of the parameters provides better results for CRP than the original values given by the authors for the similar problems (CP and $GCSP$). For this reason, in this section we study the behavior of the different parameters of the proposed heuristics, in order to select the best option for our problem.

4.1 Design of experiments

In order to calibrate the algorithms, we apply a design of experiments (Montgomery, 2005). We have chosen a full factorial design in which all possible combinations of the following factors are tested for each method:

- IG has two parameters: δ and T. We test the same range of values as in Ruiz and Stützle (2007), i.e.:
  - $T \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5\}$

  - $\delta \in \{2, 3, 4, 5, 6, 7, 8\}$

  Both δ and T were already calibrated by Ruiz and Stützle (2007) for CP obtaining the best results for $\delta = 4$ and $T = 0.4$. Nevertheless, their design of experiments show that both parameters are very robust, that means, that in the statistical analysis developed by the authors, the different levels of both parameters do not yield significative differences, suggesting that the heuristic is rather robust with respect to them.

- SGAT has five parameters: $size_{population}$, $pressure$, p_C, p_M and p_{LS}. We test the same range of values as in Ruiz and Allahverdi (2009), i.e.:

  - $size_{population} \in \{30, 50, 70\}$
  - $pressure \in \{10, 30, 50\}$
  - $p_C \in \{0.3, 0.5, 0.7\}$
  - $p_M \in \{0.01, 0.02, 0.03\}$
  - $p_{LS} \in \{0.1, 0.15, 0.2\}$

  In Ruiz and Allahverdi (2009) the best values for the $GCSP$ were found by setting the population size to 50 individuals, the pressure parameter to 30%, $p_C = 0.3$, $p_M = 0.02$, and $p_{LS} = 0.15$.

- Finally, RVNS has two parameters: k_{max} and q. q is randomly generated in a given interval of n_O or n_N, so the levels for this parameter are intervals. We test the following values:

  - $k_{max} \in \{10, 20, 30, 40, 50, 60\}$
  - $q \in \{[25, 50], [50, 75], [75, 95]\}$

All the cited factors result in a total of $6 \times 8 = 48$ different combinations and consequently 48 different IG algorithms, $3^5 = 243$ SGATs and $6 \times 3 = 18$ RVNSs. We refer to each of these algorithms as versions.

4.2 Test-bed design

Each version identified in the previous subsection is tested using the test-bed by Taillard (1993). This test-bed is well-known for serving as benchmark for scheduling problems, and consists of 120 instances of various sizes $n \times m$, with 10 instances for each size and $n \in \{20, 50, 100, 200, 500\}$ and $m \in \{5, 10, 20\}$.

For our problem, we need two set of jobs, J_O and J_N. For this reason we use the processing times available for each instance as the processing times for both sets, being $n_O = n/2$ and $n_N = n/2$. We have selected the same size for both sets based on a preliminary study carried out to analyse the structure of optimal solutions for small instances, which is not presented in this work due to space limitations. For further details, we refer to the reader to Perez-Gonzalez (2009). So the data n_O, n_N (since $n = n_O + n_N$), m and the processing times are provided by the test-bed.

In addition, a common due date for jobs in J_O must be generated for each instance. Although there are several methods to generate common due dates in the literature, we need a suitable method for which rescheduling makes sense. On one hand, a tight common due date with respect to the makespan of J_O will not allow rescheduling them together with J_N, since changing the schedule of jobs in J_O would most likely lead to unfeasible schedules. Instead the schedule of jobs in J_O will remain fixed, and the problem will turn into a machine availability problem. On the other hand, a loose common due date for J_O would not be realistic, and the due date will be verified for any schedule regarding the sequence of jobs in J_O, so the problem will turn into a classical permutation flowshop problem. In Perez-Gonzalez (2009), existing methods in the literature for generating due dates, including the ones by Sarper (1995); Sakuraba et al. (2009); Armentano and Ronconi (1999); Gelders and Sambandam (1978); Hasija and Rajendran (2004), and a method adapted from Unal et al. (1997) for a rescheduling problem are analysed. From this analysis, it turns out that the latter method serves to provide the most realistic due dates as compared to the other methods. This method consists of generating d according to the distribution $d \sim U[C_{max}^{J_O}, C_{max}^{J_O}(1 + R)]$ with R a slack factor greater than zero, similar to the idea suggested by Unal et al. (1997). More specifically, the best results are obtained for $R = 0.4$ and $C_{max}^{J_O}$ as the makespan provided by the NEHT applied to J_O.

Therefore, these values are employed in the subsequent computational experience.

4.3 Results

In order to allow for a fair comparison, all heuristics have the computation time as stopping criterion. The maximum CPU time allowed is given by the expression $n\cdot(m/2)\cdot t$ milliseconds, which is used by Ruiz and Stützle (2007) and Ruiz and Allahverdi (2009) for IG and SGAT, respectively. Ruiz and Allahverdi (2009) test some values of t for SGAT, obtaining the best results for $t = 60$. This is also the value used by Ruiz and Stützle (2007) for the IG. Therefore, we will compare IG and SGAT in their best scenario regarding the CPU times.

Each problem instance in the Taillard's testbed has been solved by the adaptation of SGAT and IG, and by RVNS. To compare them, the corresponding Relative Percentage Deviation (RPD) is computed as follows:

$$RPD = \frac{C_{max}^{J_N}(HEUR) - C_{max}^{J_N}(BEST)}{C_{max}^{J_N}(BEST)} \cdot 100$$

where $C_{max}^{J_N}(HEUR)$ is the makespan obtained by heuristic $HEUR$ and the best known makespan for each instance is $C_{max}^{J_N}(BEST)$. Since there is no benchmark for our instances, this $C_{max}^{J_N}(BEST)$ value is the best among all heuristics tested.

A full factorial experiment is carried out for each heuristic, in total three experiments, with the response variable the RPD, and the parameters of each heuristic as factors, by means of a multi-factor Analysis of Variance (ANOVA) (Montgomery, 2005). Due to space limitations, we do not detail the statistical analysis process and we only summarise the results. The detailed analysis is available from the authors upon request.

From the analysis of our adaptation of the algorithm IG, we conclude that both factors, T and δ have influence on the response variable. For T, there are not statistical differences among the all levels, except $T = 0.0$ which provides the worst value of the average relative percentage deviation (Average RPD, $ARPD$). For this reason, we select $T = 0.4$, the same value that in Ruiz and Stützle (2007) in their original paper. For δ, there are not differences among $\delta = 2, 3, 4$ and 5, being this set different from $\delta = 6, 7$ and 8. The best values for $ARPD$ are provided by the levels of the first set (2, 3, 4 and 5). Accordingly, we select $\delta = 4$, the same value that in Ruiz and Stützle (2007) in the original paper.

Results for SGAT reveal that our adaptation of the heuristic is robust for $size_{population}$, p_C y p_{LS}. There are not statistically significative differences among the levels of these parameters, so we select for our adaptation the same values that in Ruiz and Allahverdi (2009) in their original paper. The $pressure$ factor has influence on the RPD, providing the best value for $pressure = 30$. In the case of p_M there are not statistically significative differences between $p_M = 0.03$ and 0.02, selecting the same value that Ruiz and Allahverdi (2009), i.e. $p_M = 0.02$.

Finally, the analysis shows that RVNS is robust for both parameters, k_{max} and q, providing the best results for $k_{max} = 40$ and q randomly generated between 75% and 95% of n_O or n_N.

As a summary, the levels for each factor are shown in Table 3, where the bold figures indicate those finally selected. Once the three algorithms have been calibrated, the comparison among them is described in the next section.

Heuristic	Factors	Levels
IG	T	0.0, 0.1, 0.2, 0.3, **0.4**, 0.5
	δ	2, 3, **4**, 5, 6, 7, 8
SGAT	$size_{population}$	30, **50**, 70
	$pressure$	10, 20, **30**
	p_C	**0.3**, 0.5, 0.7
	p_M	0.01, **0.02**, 0.03
	p_{LS}	0.1, **0.15**, 0.2
RVNS	k_{max}	10, 20, 30, **40**, 50, 60
	q	[25,50], [50,75], **[75,95]**

Table 3: Parameter values used for calibration of the heuristics (selected values are in bold).

5 Computational experience

In order to evaluate the effectiveness of the RVNS and the two heuristics adapted from the literature, we carry out an extensive computational analysis. We employ the test-bed by Taillard (1993). Note that the aim of the calibration was not to check the efficiency of IG and SGAT for all instances, but to compare them to RVNS in the most favourable conditions. Therefore, both calibration and the experimental test-bed are the same. The parameters used for the three algorithms are those selected in the previous section. The computation time is also employed as stopping criterion, and the corresponding relative percentage deviation (RPD) is computed.

30 independent trials have been run for each instance, and for each algorithm, we tried to perform an analysis of variance to validate the statistical significance of the observed differences in the quality of the solutions. The type of heuristic is considered as factor with three levels (SGAT, IG and RVNS). The dependent variable is the RPD obtained for each instance. Thus, we have $30 \cdot 120$ observations for each heuristic, 10,800 observations in total. The convention in most research is to use a significance level of 0.05, so we employ it for all statistical tests developed. The Levene test is applied to check the homocedasticity condition. The p-values obtained for all statistics of central tendency are lower than 0.05, so we reject the null hypotheses about homogeneity of variance. Therefore, applying analysis of variance is not suitable in this case, so we apply non-parametric tests. The p-values obtained by the non-parametric tests (Kruskal-Wallis and Mann-Whitney) are lower than 0.05 indicating that all the pairwise differences are statistically different in mean. Therefore, we can state that the three heuristics are different regarding the RPD. The Average of RPD ($ARPD$) over all trials for a problem size is computed. The results for the three heuristics are given in Table 4, and show that IG provides the highest values of $ARPD$ for all instance sizes. This bad result can be explained because IG spends a long time in checking the feasibility of all solutions involved in the process. SGAT and RVNS provide better results, being RVNS the best for all 120 instances. The most remarkable differences are obtained for the smallest problems, i.e. $n_O \times n_N = 10 \times 10$ with an average of 2.1873 for SGAT and 0.6845 for RVNS.

Figure 2 shows the means and confidence intervals (at 95% confidence) for the RPD of the three heuristics. As it can be seen, the differences between SGAT and RVNS are statistically significant.

Moreover, in order to test the robustness of RVNS for different CPU times, the be-

n_O	n_N	M	**IG**	**SGAT**	**RVNS**
10	10	5	3.0412	0.2445	0.1632
		10	8.6571	2.9801	1.7056
		20	12.0777	3.3373	0.1845
		Subtotal	7.9253	2.1873	0.6845
25	25	5	2.6794	0.1137	0.0643
		10	7.8231	0.9070	0.6162
		20	9.5048	1.2102	0.8183
		Subtotal	6.6691	0.7436	0.4996
50	50	5	2.0026	0.0707	0.0600
		10	6.9777	0.4231	0.3817
		20	9.3208	0.9473	0.3740
		Subtotal	6.1004	0.4804	0.2719
100	100	10	4.8322	0.1415	0.1932
		20	7.9620	0.8429	0.4104
		Subtotal	6.3971	0.4922	0.3018
250	250	20	4.9103	0.3900	0.2796
Average			6.6491	0.9674	0.4376

Table 4: *ARPD* for IG, SGAT and RVNS.

haviour of the heuristics is checked for $t = 30$ and $t = 120$. Table 5 shows the $ARPD$ values obtained for all heuristics for each case. As it can be seen, RVNS is the best for all tested CPU times, maintaining similar differences with SGAT for all cases.

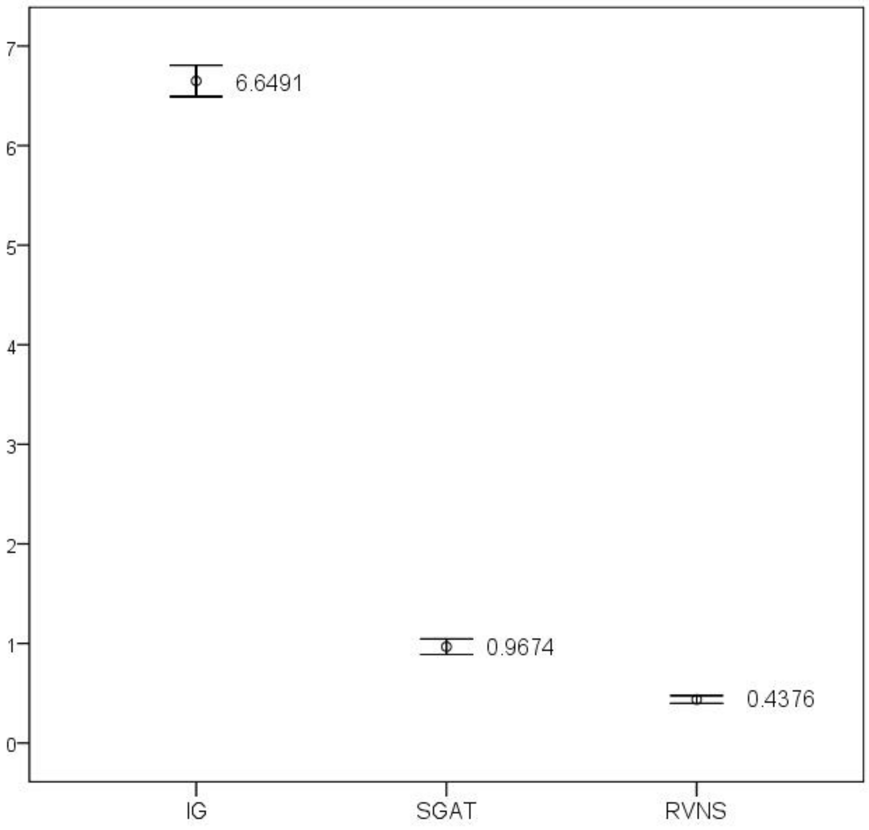

Figure 2: Means and 95% LSD intervals for IG, SGAT and RVNS.

t	**IG**	**SGAT**	**RVNS**
30	7.2952	1.0603	0.6943
60	6.6249	0.9448	0.4986
120	6.3124	0.7218	0.4351

Table 5: $ARPD$ for different values of t.

6 Conclusions

This paper aims at a special case of a rescheduling problem, which is motivated by the need of setting a common due date for a set of jobs while there is another set of jobs in the system that have been previously scheduled. The problem is denoted as CRP, and it is NP-hard in the strong sense for more than two machines.

To solve it, we have developed a new heuristic for the problem based on VNS, called Refreshing Variable Neighbourhood Search (RVNS). This heuristic is capable to handle both feasible and unfeasible solutions. We avoid penalising the objective function by introducing the concept of strict and relaxed solutions, simplifying the original objective function to $C_{max}^{J_N}$ which makes easier to compare the solutions, and to check the feasibility only for those sequences identified as good ones by the heuristic.

To the best of our knowledge, CRP is a new problem and it is not possible to compare the RVNS proposed with existing algorithms. To show the efficiency of our proposal we are constrained to searching for similar problems, analysing the best existing heuristics for these problems, adapting them to our problem (and at the same time maintaining most of the original nature of these heuristics). Then, two heuristics have been adapted: the IG algorithm by Ruiz and Stützle (2007) to solve the classical permutation flowshop problem (CP), and the SGAT by Ruiz and Allahverdi (2009) for the general constrained scheduling problem ($GCSP$), and consequently for the constrained scheduling problem (CSP). Trying to conduct the fairest possible computational experience, we have carried out a calibration of the parameters of SGAT and IG in order to be sure that these algorithms are tested under their most favourable conditions. After this calibration, we determine the best combination of parameters for the three algorithms, which for IG and SGAT are the same as in the original problems.

The computational experience carried out shows that RVNS is statistically different and better than SGAT, and that the IG algorithm, which is very efficient for the unconstrained problem, exhibits a poor performance for our problem. The effectiveness of RVNS is also tested for different CPU times. As the differences between SGAT and our proposal are not that pronounced as with IG, and since CSP can be seen as a sort of generalisation of our problem, we can conclude that both problems are similar not only from a theoretical point of view, but also from a practical viewpoint.

To the best of our knowledge, this is the first time that VNS is applied to constrained flowshop problems. Since the heuristic makes use of specific knowledge about constrained problems, it is possible to extend it to other problems of this nature, considering the wide literature about constrained problems in scheduling, for example in permutation flowshop problems (see e.g., Minella et al., 2008).

As a future research line, several related rescheduling problems can be identified: for example, allowing the completion of the jobs in J_O in an interval ϵ, i.e., considering the objective $C_{max}^{J_N}/T_{max}^{J_O} \leq \epsilon$; or penalising the tardiness by a linear function of $C_{max}^{J_N}$ and $T_{max}^{J_O}$. Moreover, the case with different due dates for the jobs in J_O and J_N implies different constraints and objectives. Tight due dates for jobs in J_N would be guaranteed by the minimisation of the flowtime, and the fulfilment for the due dates of J_O by the constraint on the total tardiness. These problems may be compared to the problems studied here, applying the heuristics presented in this paper to solve them, or even developing more sophisticated methods based on these heuristics. Finally, the addressed scheduling problem may be applied to other scheduling environments such as single or parallel machine environments, e.g., Damodaran et al. (2009), Chen et al. (2009). Also, in order to minimize the total cost of inventory (e.g., see Singh and Chand, 2009, Warburton, 2009, Sharma, 2009) other objective functions that take into account factors to minimize the cost of inventory may be investigated.

Acknowledgements

The authors wish to thank the anonymous referees for their thorough and ameliorating review as well as for their helpful suggestions. The work of the authors has been supported by CICYT Project DPI2007-61345 funded by the Spanish Ministry of Science and Innovation, and grant P08-TEP-03630 funded by the Andalusian Government.

References

Allahverdi, A. (2004). A new heuristic for m-machine flowshop scheduling problem with bicriteria of makespan and maximum tardiness. *Computers & Operations Research*, 31(2):157–180.

Armentano, V. A. and Ronconi, D. P. (1999). Tabu search for total tardiness minimization in flowshop scheduling problems. *Computers & Operations Research*, 26(3):219–235.

Blazewicz, J., Pesch, E., Sterna, M., and Werner, F. (2008). Metaheuristic approaches for the two-machine flow-shop problem with weighted late work criterion and common due date. *Computers & Operations Research*, 35(2):574–599.

Breit, J. (2004). An improved approximation algorithm for two-machine flow shop scheduling with an availability constraint. *Information Processing Letters*, 90(6):273–278.

Chakravarthy, K. and Rajendran, C. (1999). A heuristic for scheduling in a flowshop with the bicriteria of makespan and maximum tardiness minimization. *Production Planning & Control*, 10(7):707–714.

Chen, Y., Li, X., and Sawhney, R. (2009). Restricted job completion time variance minimisation on identical parallel machines. *European Journal of Industrial Engineering*, 3(3):261–276.

Cheng, T. C. E. and Jiang, J. (1998). Job shop scheduling for missed due-date performance. *Computers & Industrial Engineering*, 34(2):297–307.

Damodaran, P., Hirani, N. S., and Velez-Gallego, M. C. (2009). Scheduling identical parallel batch processing machines to minimise makespan using genetic algorithms. *European Journal of Industrial Engineering*, 3(2):187–206.

Daniels, R. L. and Chambers, R. J. (1990). Multiobjective flow-shop scheduling. *Naval Research Logistics*, 37(6):981–995.

Framinan, J. M. (2007). Scheduling for due date fulfillment: a computational study. European Conference on Operational Research, EURO XXII, Prague, Czechoslovakia, page 209.

Framinan, J. M. (2008). Managing resources for order promising in available-to-promise (ATP) systems: A simulation study. Proceedings of the International Conference on Flexible Automation and Intelligent Manufacturing (FAIM 2008). Skövde (Sweden).

Framinan, J. M. (2009). Handling variability for robust order promising and fulfilment. In *Computers & Industrial Engineering, 2009. CIE 2009. International Conference on*, pages 714–719.

Framinan, J. M., Gupta, J. N. D., and Leisten, R. (2004). A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society*, 55(12):1243–1255.

Framinan, J. M. and Leisten, R. (2006). A heuristic for scheduling a permutation flowshop with makespan objective subject to maximum tardiness. *International Journal of Production Economics*, 99(1):28–40.

Framinan, J. M. and Leisten, R. (2007). Total tardiness minimisation in permutations flow shops: a simple approach based on a variable greedy algorithm. *International Journal of Production Research*, 46(22):6479–6498.

Framinan, J. M. and Leisten, R. (2009). Available-to-promise (ATP) systems: a classification and framework for analysis. *International Journal of Production Research*, In press.

Garey, M. R., Johnson, D. S., and Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129.

Gelders, L. F. and Sambandam, N. (1978). Four simple heuristics for scheduling a flowshop. *International Journal of Production Research*, 16(3):221–231.

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

Hall, N. G. and Potts, C. N. (2004). Rescheduling for new orders. *Operations Research*, 52(3):440–453.

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

Hasija, S. and Rajendran, C. (2004). Scheduling in flowshops to minimize total tardiness of jobs. *International Journal of Production Research*, 42(11):2289–2301.

Lee, C. Y. (1991). Parallel machines scheduling with nonsimultaneous machine available time. *Discrete Applied Mathematics*, 30(1):53–61.

Lourenco, H. R., Martin, O., and Stützle, T. (2002). Iterated local search. In Glover, F. and Kocheberger, G. A., editors, *Handbook of Metaheuristics*, pages 321–353. Kluver Academic Publisher, Norwell, MA.

Michalewicz, Z. (1994). *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, Berlin (Germany), second edition.

Minella, G., Ruiz, R., and Ciavotta, M. (2008). A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. *INFORMS Journal on Computing*, 20(3):451–471.

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

Mocquillon, C., Lenté, C., and T'kindt, V. (2008). Rescheduling for new orders with setup times. Eleventh International Workshop on Project Management and Scheduling, PMS 2008. Istanbul (Turkey), pages 203–205.

Montgomery, D. C. (2005). *Design and analysis of experiments*. John Wiley and Sons, New York (USA), 6th edition.

Nawaz, M., Enscore Jr, E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega, The International Journal of Management Science*, 11(1):91–95.

Osman, I. H. and Potts, C. N. (1989). Simulated annealing for permutation flow-shop scheduling. *Omega, The International Journal of Management Science*, 17(6):551–557.

Pan, Q. K., Fatih Tasgetiren, M., and Liang, Y. C. (2008). A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. *Computers & Operations Research*, 35(9):2807–2839.

Perez-Gonzalez, P. (2009). *Common due date setting in permutation flowshops: Analysis of problems and solutions procedures*. PhD thesis, Industrial Management Research Group, University of Seville (Spain).

Perez-Gonzalez, P. and Framinan, J. M. (2009). Scheduling permutation flowshops with initial availability constraint: Analysis of solutions and constructive heuristics. *Computers & Operations Research*, 36(10):2866–2876.

Philipoom, P. R. (2000). The choice of dispatching rules in a shop using internally set due-dates with quoted leadtime and tardiness costs. *International Journal of Production Research*, 38(7):1641–1655.

Ragatz, G. L. and Mabert, V. A. (1984). A simulation analysis of due date assignment rules. *Journal of Operations Management*, 5(1):27–39.

Rangsaritratsamee, R., Ferrell, W. G., and Kurz, M. B. (2004). Dynamic rescheduling that simultaneously considers efficiency and stability. *Computers & Industrial Engineering*, 46(1):1–15.

Ruiz, R. and Allahverdi, A. (2007). No-wait flowshop with separate setup times to minimize maximum lateness. *The International Journal of Advanced Manufacturing Technology*, 35(5):551–565.

Ruiz, R. and Allahverdi, A. (2009). Minimizing the bicriteria of makespan and maximum tardiness with an upper bound on maximum tardiness. *Computers & Operations Research*, 36(4):1268–1283.

Ruiz, R. and Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(3):479–494.

Ruiz, R., Maroto, C., and Alcaraz, J. (2006). Two new robust genetic algorithms for the flowshop scheduling problem. *Omega, The International Journal of Management Science*, 34(5):461–476.

Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 117(3):2033–2049.

Sakuraba, S., Ronconi, D. P., and Sourd, F. (2009). Scheduling in a two-machine flowshop for the minimization of the mean absolute deviation from a common due date. *Computers & Operations Research*, 36(1):60–72.

Sarper, H. (1995). Minimizing the sum of absolute deviations about a common due date for the two-machine flow shop problem. *Applied Mathematical Modelling*, 19(3):153–161.

Sharma, S. (2009). On price increases and temporary price reductions with partial backordering. *European Journal of Industrial Engineering*, 3(1):70–89.

Singh, O. P. and Chand, S. (2009). Supply chain design in the electronics industry incorporating JIT purchasing. *European Journal of Industrial Engineering*, 3(1):21–44.

Taillard, E. D. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74.

Taillard, E. D. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285.

T'kindt, V. and Billaut, J. C. (2006). *Multicriteria scheduling: Theory, models and algorithms*. Springer, Berlin (Germany), second edition.

Turner, S. and Booth, D. (1987). Comparison of heuristics for flow shop sequencing. *Omega, The International Journal of Management Science*, 15(1):75–78.

Unal, A. T., Uzsoy, R., and Kiran, A. S. (1997). Rescheduling on a single machine with part-type dependent setup times and deadlines. *Annals of Operations Research*, 70:93–113.

Vig, M. M. and Dooley, K. J. (1991). Dynamic rules for due-date assignment. *International Journal of Production Research*, 29(7):1361.

Warburton, R. D. H. (2009). EOQ extensions exploiting the lambert w function. *European Journal of Industrial Engineering*, 3(1):45–69.

Yang, B. (2007). Single machine rescheduling with new jobs arrivals and processing time compression. *The International Journal of Advanced Manufacturing Technology*, 34(3-4):378–384.