3rd International Conference on Industrial Engineering and Industrial Management
XIII Congreso de Ingeniería de Organización
Barcelona-Terrassa, September 2nd-4th 2009

Comparison of heuristics for rescheduling in permutation flowshops*
Comparación de heurísticas en resecuenciación en flujo regular de permutación

Paz Pérez-González, José M. Framiñán Torres, José M. Molina Parente y José L. Andrade Pineda
Departamento de Organización Industrial y Gestión de Empresas. Escuela Técnica Superior de Ingenieros.
Universidad de Sevilla. Avda. de los Descubrimientos, s/n. 41092 Sevilla.
pazperez@esi.us.es *framinan@us.es*

Fecha de recepción: 08-09-09
Fecha de aceptación: 15-09-09

Abstract. Rescheduling is one of the main consequences of the variability in the shop floor, as a number of unforeseeable disruptions make impossible to follow the original schedule. In this paper we study a flowshop rescheduling problem where a set of jobs arrives to the system and it is scheduled together with jobs already present. The objective is to minimise the makespan of the new jobs and constrained with the fact that the maximum tardiness of the old jobs must be equal to zero. The problem is NP-hard, so we compare heuristic methods in order to select the best option.

Key words: scheduling, flowshop, makespan, rescheduling, heuristics.

Resúmen. La resecuenciación es una de las principales consecuencias de la variabilidad en los talleres, ya que las interrupciones no hacen posible el seguimiento de la secuencia inicial. Este trabajo aborda un problema de resecuenciación en flujo regular donde un conjunto de trabajos llega al sistema, y son secuenciados junto con un conjunto de trabajos que ya está planificado. El objetivo es minimizar el makespan de los nuevos trabajos, restringido a que la máxima tardanza de los trabajos antiguos debe ser cero. El problema es NP, por lo que se comparan diferentes métodos heurísticos para su resolución, y se selecciona la mejor opción.

Palabras clave: secuenciación, flujo regular, makespan, resecuenciación, heurísticas.

1. Introduction

Rescheduling is one of the main consequences of the variability in the shop floor, as a number of unforeseeable disruptions make impossible to follow the original schedule (Wu *et al.,* 1993). Typical disruptions are the arrival of new orders, order cancellations, rush orders, etc. (Hall and Potts, 2004). In the case of the arrival of new jobs at the time that the current schedule is being executed, there are two broad options for the scheduler: a) considering the jobs in the system as «frozen» (i.e. the schedule of the previous jobs is maintained) or b) allowing the modification of the schedule of the jobs in the system and rescheduling them together with the new jobs. The first approach implies a machine availability constraint problem, denoted MACP, studied by Pérez González and Framiñán (2009). The second approach is addressed in this work.

2. Problem statement

In line with the related literature (see e.g. Unal *et al.,* 1997), jobs are classified either as 'old', or 'new' jobs. We assume that the set of old jobs, denoted J_O with n_O jobs, belongs to a previously scheduled order, so they share a common due date d which is a given parameter. A set of new jobs, J_N with n_N jobs, arrives to the system. The proposal of the problem under consideration —called RP in this work— is to determine the schedule S formed by jobs belonging to $J = J_O \cup J_N$ with $n = n_O + n_N$ jobs, and objective the minimisation of the makespan of new jobs, denoted

* This work stems from the participation of the authors in the research project «Advanced Systems for Integrated Order Management», grant DPI2007-61345, funded by the Spanish Ministry of Science and Innovation, and «SCOPE», grant P08-TEP-3630, funded by the Andalusian Government.

as $C^{J_N}_{\max}(S)$, in order to set a tight due date for the new jobs while not violating the due date of the old jobs. 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, i.e. $T^{J_O}_{\max}(S) = \max T^{J_O}_j(S) = 0$, with $T^{J_O}_j(S) = \{0, C^{J_O}_j(S) - d\}$ the tardiness of the job $j \in J_O$, and $C^{J_O}_i(S)$ the completion time of the job $j \in J_O$ in the last machine. Note that it is 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.

Among the different shop floor layouts, we focus on the flowshop. Using the notation defined by Graham *et al.* (1979), our problem 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 $C^{J_N}_{\max} / T^{J_O}_{\max} = 0$ the constrained objective function.

This problem is strongly NP-hard for more than two machines, since if we consider the set of old jobs as an empty set then it is reduced to the classical permutation flowshop scheduling problema with makespan objective (the classical problem is denoted as CLP), which is known to be strongly NP-hard (Pinedo, 1995). For this reason heuristic methods are proposed in this paper in order to solve the problem in an approximate way.

2.1. Related literature

To the best of our knowledge, RP has not been as we consider two different objectives, one for each set of jobs. Only some references consider similar problems for a single machine, for example, Unal *et al.* (1997), Hall and Potts (2004), Mocquillon *et al.* (2008) and Yang (2007). There are similar approaches in the literature on flowshop scheduling with due date related objectives: the permutation flowshop problem with the objective of minimising the makespan subject to a given maximum tardiness, denoted $Fm|prmu|\varepsilon(C_{\max}, T_{\max})$ following the notation of T-kindt and Billaut (2002) for multi-criteria scheduling problems. For convenience, we denote this constrained scheduling problem as COP. It is a special case of a more general problem denoted $Fm|prmu|\varepsilon(Z, T_{\max})$ with $Z = \lambda C_{\max} + (1-\lambda)T_{\max}$, $\lambda \in [0,1]$. We denote this problem as Generalised COP or GCOP, being COP the case when $\lambda = 1$. Given the similarity of these problems, the methods applied to GCOP or COP could be adapted to our problem RP. The best method to solve GCOP, and consequently to COP, is provided by Ruiz and Allahverdi (2009).

3. Heuristic methods

Solving the rescheduling problem under consideration is not an immediately issue. As mentioned before, the problem is NP-hard, and its comparison with the CLP and the MACP, reveals that RP is the most difficult one (Pérez González *et al.,* 2007). We have adapted the best existing methods found in the literature for the most similar problems to our problem (CLP and COP) in order to compare them with a new proposed heuristic. The Iterated Greedy (IG) algorithm proposed by Ruiz and Stützle (2007) is among the best methods to solve the CLP. Currently, the best heuristic known to solve COP is the so-called Steady State Genetic Algorithm, SGAT, proposed by Ruiz and Allahverdi (2009). We have adapted both methods to the problem under consideration and proponed a new heuristic based on the Variable Neighbourhood Search (VNS), developed by Mladenovic and Hansen (1997). The heuristics and results are detailed in the following subsections.

3.1. Iterated Greedy algorithm

Iterated Greedy (IG) algorithm (Ruiz and Stützle, 2007) is a heuristic method which generates a sequence of solutions by iterating over greedy constructive heuristics. It is simple and easily applicable to other problems, so the adaptation to our problem implies a minor modification, maintaining the original values of the parameters. We consider feasible solutions, considering $C^{J_N}_{\max}(S)$ as objective function for each solution S. The pseudo-code is shown in Figure 1.

In the original IG, the initial sequence is constructed by NEHT, improved version by Taillard (1990) of the NEH heuristic (Nawaz *et al.,* 1983). For our problem, the initial solution has to be feasible and it is generated by the Initial Feasible Solution method which uses an adaptation of the NEHT to the MACP, called ANEHT, considering the machine availability instants (a_i) to compute the makespan. The Destruction and Construction procedures are detailed in Ruiz and Stützle (2007). The parameter δ associated to these procedures is adjusted obtaining the best results for $\delta = 4$, being very robust. The local search procedure, called Iterative Improvement, improves each solution generated in the construction phase. The last step of IG 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, with a constant temperature, $Temperature = T \cdot \sum_{i=1}^{m} \sum_{j=1}^{m} p_{ij} / n \cdot m \cdot 10$, where T is a

Figure 1
Pseudo-code of IG

```
procedure Iterated_Greedy
    % STEP 1:  Initialisation
    $S$ := Initial_Feasible_Solution;
    $S$ := Iterative_Improvement($S$);
    $S^{best}$ := $S$;
    while (termination criterion not satisfied) do
    % STEP 2:  Destruction
        $S'$ := $S$;
        for $k = 1$ to $\delta$ do
            $S^D$ := remove one job at random from $S'$ and insert it in $S^R$;
        end for
    % STEP 3:  Construction
        for $k = 1$ to $\delta$ do
            $S'$ := best permutation obtained by inserting job $S_k^R$ in all
            possible positions of $S^D$;
        end for
    % STEP 4:  Local Search
        $S''$ := Iterative_Improvement($S'$);
    % STEP 5:  Acceptance Criterion
        if $C_{max}(S'') < C_{max}(S)$ then
            $S$ := $S''$;
            if $C_{max}(S) < C_{max}(S^{best})$ then
                $S^{best}$ := $S$;
            end if
        else if $(random \leq \exp -(C_{max}(S'') - C_{max}(S))/Temperature)$ then
            $S$ := $S''$;
        end if
    end while
    return $S_{best}$;
end
```

parameter set to $T = 0.4$, being the heuristic rather robust with respect to T.

3.2. Steady State Genetic Algorithm

In the Steady State GA, presented by Ruiz and Allahverdi (2009), 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. Only minor modifications have to be implemented to adapt SGAT to our problem, maintaining the original values of the parameters. SGAT is described in Figure 2.

In the original SGAT, two super-individuals are generated in the first step, one by the EDD (Earliest Due Date) rule and the other by the NEHT method. 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 NEHT to $J = J_O \cup J_N$. The second super-individual is generated by the Initial Feasible Solution method previously described for IG. The population size selected by Ruiz and Allahverdi (2009) consists of 50 individuals. SGAT allows unfeasible solutions in the population by defining three states for the population. The fitness value of each individual may be calculated depending on each state. In State 1, all

Figure 2
Pseudo-code of SGAT

```
procedure SGAT
    % STEP 1:  Initialisation
    $S^1$ := NEHT($J$);
    $S^2$ := Initial_Feasible_Solution;
    insert $S^1$ and $S^2$ into population;
    for $k = 3$ to $size_{population}$ do
        $S^k$ := randomly generated sequence;
        insert $S^k$ into population;
    end for
    for $k = 1$ to $size_{population}$ do
        $F^k$ := Fitness($S^k$, $population$);
    end for
    % STEP 2:  New Population
    while (termination criterion not satisfied) do
        for $k = 1$ to 2 do
            select $S^k$ by n_tournament(population);
        end for
        TP_Crossing($S^1$,$S^2$);
        Shift_Mutation($S^1$,$S^2$);
        for $k = 1$ to 2 do
            LightLS($S^k$);
            $F^k$ := Fitness($S^k$, $population$);
            $uniqueness$= Uniqueness($S^k$, $population$);
            if($uniqueness = true$) then
                $F^{worst}$ := Fitness($S^{worst}$, $population$);
                if($F^k < F^{worst}$) then
                    remove $S^{worst}$ from population;
                    insert $S^k$ in population;
                end if
            end if
        end for
    end while
end
```

solutions are feasible. For this state, the objective value (C^N_{max} in our problem) of the individual is considered as the fitness value. In State 2, there is a mixture of unfeasible as well as feasible solutions in the population. In this case, the fitness value is computed by calculating the worst feasible objective value and by adding this value to that of the unfeasible solutions. Thus, the best unfeasible solution always has a fitness value that is worse than the worst feasible solution. In State 3, all solutions are unfeasible. The fitness value considers the maximum tardiness given by the sequence. By doing so, the SGAT is pushed so that feasible solutions are found as fast as possible.

For parent selection, a fast and simple selection operator, the n-tournament selection procedure, is chosen. They set the pressure parameter to 30%. The crossover procedure selected is the two-point (TP) (see Michalewicz, 1994). The probability of carrying out a crossover after selection is called p_C, and the best value tested by Ruiz and Allahverdi (2009) is $p_C = 0.3$. The mutation operator consists of extracting one job from the individual and re-inserting it in another random position. The probability of an individual to be mutated is p_M, set to 0.02. SGAT also incorporates a Light Local Search which is applied to the best solution in the initial population, and to a fraction p_{LS}, set to 0.15, of offsprings generated after the crossover and mutation. We have adapted the Taillard's improvement to speed up the Light Local Search for RP, by discarding unfeasible positions if the job involved in the insertion belongs to J_O. Finally, each new individual is checked to guarantee its «uniqueness» once the fitness has been calculated in order to avoid clones in the population. These procedures are detailed in Ruiz and Allahverdi (2009).

3.3. Refreshing VNS

Variable Neighbourhood Search (VNS) was developed by Mladenovic and Hansen (1997). It is a metaheuristic based on changing the neighbourhood in a local search procedure. 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. For this reason, our variant of VNS tries to repair the solutions (similarly to the idea proposed by Allahverdi, 2004), and includes an escape method. In addition, we define two types of solutions: a) strict solutions, with the feasibility guaranteed for this kind of solution, and they can be a solution for the constrained problem; and b) relaxed solutions, which can be either feasible or unfeasible. Relaxed solutions are considered for the construction of neighbourhood structures. In general, they cannot be solutions for our problem. Only some relaxed solutions are selected in the heuristic to check its feasibility. In this case, if a relaxed solution is feasible, it is considered as strict solution. Usually, when unfeasible solutions are considered, the objective function is penalised. However, as we assume relaxed and strict solutions, the same measure is defined as objective for both kinds of solutions. Figure 3 shows the pseudo-code of the Refreshing VNS.

Our method starts with two sequences, S_{best}, which is generated by the Initial Feasible Solution procedure explained previously and compared to the new strict solutions found by the method, and the initial relaxed solution S obtained by the NEHT applied to all jobs in J. If S is feasible and better than S_{best}, then $S_{best} = S$.

A 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. We set $k_{max} = n*20\%$, as it makes no sense to explore very large neighbourhoods of S once smaller neighbourhoods have been explored without good results. 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_0, if it is feasible and better than S_{best}, the latter is updated. Then, if S_0 is better than S, then a boolean flag change is set as true. In this case the local search is applied. Otherwise, the escape method is carried out.

Several local search procedures are applied to the relaxed sequence after the shaking in the case that change equals true. The General Local Search is an iterative improvement method where a job selected at random is 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. If the relaxed solution obtained by the General Local Search is feasible, the Intensify method tries to improve it. A special case occurs when the makespan of the jobs in J_N is lower than the common due date of J_O, i.e. all jobs are sequenced before the due date. An iterative improvement method (denoted as Special Case) is built for this situation. 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 described in the previous subsections cannot be applied. Instead, an Escape method allows escaping from 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 between 75% and 90% of n_N or n_N, depending on the feasibility of the sequence. If it is feasible, $q \leq n_N$ 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, $q \leq n_O$ jobs of J_O are removed and inserted by the same

Figure 3
Pseudo-code of RVNS

```
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 or 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
```

procedure. For both cases, if the sequence obtained is feasible and better than S_{best}, then S_{best} is updated.

4. Computational experience

In order to evaluate the effectiveness of the Refreshing VNS as compared to IG and SGAT, we carry out an extensive computational analysis. We employ the test-bed by Taillard (1993), which consists of 120 instances of various sizes $n \times m$, with $n = \{20,50,100,200,500\}$ and $m = \{5,10,20\}$. These instances constitute an excellent benchmark to test different solution procedures for CLP, being widely used in this context (see e.g. Ruiz and Stützle, 2007 and Framinan and Leisten, 2007). We need two set of jobs, J_O and J_N. Then, we use the processing times available for each instance as the processing times for both sets, being $n_O = n/2$ and $n_N = n/2$. In addition, a common due date for jobs in J_O must be generated for each instance. A tight common due date with respect to the makespan of J_O will not allow rescheduling them together with J_N. Instead they will remain fixed, and the problem will turn into a MACP. On the other hand, a loose common due date for J_O would be not realistic, and the due date will be verified for any schedule, so the problem will turn into a CLP. Different methods for generating due dates in the flowshop literature have been analysed, turning out that generating a due date according to the distribution d~$U[C^{J_O}_{max}, C^{J_O}_{max}(1+R)]$ with R a slack factor greater than zero, similar to the idea suggested by Unal et al. (1997), serves to provide the most realistic due dates as compared to other methods. The best results have been obtained for $R=0.4$. The makespan provided by the NEHT applied to J_O is employed as value for $C^{J_O}_{max}$.

The parameters used for IG and SGAT are those in the original description of the heuristics. Moreover, in both cases the stopping criterion is the computation time, given by the expression $n*(m/2)*t$ milliseconds, with $t=60$. To compare the heuristics, the relative percentage deviation is computed by RPD= $C^{J_N}_{max}(HEUR)-C^{J_N}_{max}(BEST)/C^{J_N}_{max}(BEST)$, where $C^{J_N}_{max}(HEUR)$ is the makespan obtained by heuristic *HEUR* and the best known makespan for each instance is $C^{J_N}_{max}(BEST)$. 30 independent trials have been run for each instance. The results in Table 1 show that the IG is not suitable to solve RP, as it provides the highest values of ARPD. SGAT and RVNS provide better results, being RVNS the best with respect to the average for all 120 instances.

5. 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 RP, and it is NP-hard in the strong sense for more than two machines, so we propose several heuristics to solve it. Two of them have been adapted from similar problems in the literature, i.e. the IG algorithm by Ruiz and Stützle (2007) to solve the CLP, and the SGAT by Ruiz and Allahverdi (2009) for the GCOP. In addition, we have developed a new heuristic for the problem based on VNS, called Refreshing VNS. This heuristic is adapted to constrained problems, as it handles both feasible and unfeasible solutions. Nevertheless, we avoid penalising the objective function by introducing the concept of strict and relaxed solutions, allowing to reduce the objective function to $C^{J_N}_{max}$ in order to compare the solutions, and checking the feasibility only for the sequences identified by the heuristic as good ones. A test-bed based on the benchmark set by Taillard (1993) has been used to analyse the performance of the heuristics. The results show that RVNS is statistically different and better than SGAT, and the IG algorithm exhibits a poor performance for our problem.

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).

Table 1
ARPD for SGAT, IG and RVNS for each problem size

$n_0 \times n_N \times m$	IG	SGAT	RVNS
10×10×5	3.0412	0.2445	0.2038
10×10×10	8.6571	2.9801	1.8765
10×10×20	12.0777	3.3373	0.1965
25×25×5	2.6764	0.1137	0.0690
25×25×10	7.8093	0.8839	0.7274
25×25×20	9.4488	1.1586	1.0410
50×50×5	1.9967	0.0556	0.0519
50×50×10	6.9759	0.4213	0.4626
50×50×20	9.2723	0.9024	0.6459
100×100×10	4.8322	0.1415	0.2156
100×100×20	7.8181	0.7011	0.4656
250×250×20	4.8898	0.3572	0.2632

6. 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), pp. 157-180.

BLAZEWICZ, J., *et al.* (2008). Metaheuristic approaches for the two-machine flow-shop problem with weighted late work criterion and common due date. *Computers & Operations Research*, 35 (2), pp. 574-599.

FRAMINAN, J. M., yLEISTEN, 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), pp. 6479-6498.

GRAHAM, R., *et al.* (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5, pp. 287-326.

HALL, N.G., y POTTS, C.N. (2004). Rescheduling for new orders. *Operations Research*, 52 (3), pp. 440-453.

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

MINELLA, G.; RUIZ, R., y CIAVOTTA, M. (2008). A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. *INFORMS Journal on Computing*, 20 (3), pp. 451-471.

MLADENOVIC, N., y HANSEN, P. (1997). Variable Neighborhood Search. *Computers and Operations Research*, 24 (11), pp. 1097-1100.

MOCQUILLON, C.; LENTE, C., y T'KINDT, V. (2008). Rescheduling for new orders with setup times. Eleventh International Workshop on Project Management and Scheduling, PMS 2008, Istanbul (Turkey).

NAWAZ, M.; ENSCORE J.R., E., y 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), pp. 91-95.

PAN, Q. K.; FATIH TASGETIREN, M., y LIANG, Y. C. (2008). A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. *Computers & Operations Research*, 35 (9), pp. 2807-2839.

PÉREZ GONZALEZ, P.; FRAMIÑÁN, J.M. (2009). Scheduling permutation flowshops with initial availability constraint: analysis of solutions and constructive heuristics. *Computers & Operations Research*, 36 (10), pp. 2866-2876.

PÉREZ GONZÁLEZ, P., *et al.* (2007). Analisis de las soluciones del problema de secuenciacion en flujo regular de permutacion con maquinas no disponibles con el objetivo de minimizar el makespan. International Conference on Industrial Engineering and Industrial Management 2007, Madrid (Spain).s

PINEDO, M. (1995). *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, Englewood Cliffs, New Jersey (USA).

RUIZ, R., y ALLAHVERDI, A. (2009). Minimizing the bicriteria of makespan and maximum tardiness with an upper bound on maximum tardiness. *Computers & Operations Research*, 36 (4), pp. 1268-1283.

RUIZ, R., y STUTZLE, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 117 (3), pp. 2033-2049.

T'KINDT, V., y BILLAUT, J. C. (2002). *Multicriteria scheduling: Theory, models and algorithms*. Springer, Berlin (Germany).

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

UNAL, A.T.; UZSOY, R., y KIRAN, A.S. (1997). Rescheduling on a single machine with part-type dependent setup times and deadlines. *Annals of Operations Research*, 70, pp. 93-113.

WU, S.D.; STORER, R.H., y CHANG, P.C. (1993). One-machine rescheduling heuristics with efficiency and stability as criteria. *Computers & Operations Research*, 20 (1), pp. 1-14.

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