A New Algorithm for Solving the Single Machine Total Tardiness Scheduling Problem

DAVID ALCAIDE, JOAQUIN SICILIA
Departamento de Estadística, Investigación Operativa y Computación
Universidad de La Laguna
La Laguna, Tenerife.
SPAIN

Abstract: - We are analyzing a multifunctional machine and the set of tasks to be performed by the machine. Each task has to be finished before a given due date. We are interested in finding a schedule of the tasks in such a way that the machine complies with the due dates. The problem is formulated as a minimum total tardiness scheduling problem. An heuristic algorithm for the problem is proposed. Finally, a comparative computational experience between this algorithm and other heuristic and exact algorithms is reported.

Key-Words: - scheduling, tardiness problem, single machine, exact and heuristic algorithms CSCC99, pp.2851-2858

1 Introduction

Multifunctional machines are commonly used in industrial process. These machines are capable of performing different tasks or jobs. We can consider the machines as executing intermediate steps in the tasks performing process. The outputs of some machines are inputs for other machines. To realize the industrial process, we have to perform all the tasks involved in it. Hence, we need to plan the different tasks of each machine in such a way that the overall process is optimally completed.

The problems of deciding plans of tasks to be performed by these machines in such a manner that several objectives are achieved can usually be formalized as scheduling problems.

In this paper we consider the problem of searching for a plan to perform all the activities of a multifunctional machine in such a way that the total tardiness is minimized. With this objective the search tries to find schedules in which the machine complies with the due dates. This is important because the performed tasks must be completed for other machines and processes to continue.

Several models for the minimum total tardiness problem have been considered by different authors as Potts and van Wassenhove (1987), Schrage and Baker (1978), Lawler (1977), Lenstra and Rinnooy Kan (1980), and Blazewicz (1987). The problem has been proved to be NP-hard in the ordinary sense by Du and Leung (1990). Thus, in practice, large problems can not be optimally solved because the necessary computational time is exponential and the storage memory requirements are exponentially increased with the number of tasks. Exact methods to find optimal solutions for this problem can be seen in Baker (1974), Lawler (1977), and Potts and van Wassenhove (1982). The computational effort required by using exact methods is enormous when the number of jobs increases. In these cases, heuristic methods can provide good solutions quickly.

Wilkerson and Irwin (1971) proposed an heuristic method based on adjacent pair interchanges. Fry et al. (1989) used the best of nine adjacent pair interchange strategies and improved the quality of the solutions obtained by Wilkerson and Irwin.

Holsenback and Russell (1992) have developed an heuristic method that uses the dominance proprieties of Emmons (1969). This method improves the previous results by Fry et al. (1989) in several cases.

In this paper we also propose an heuristic algorithm for the problem and make a comparative computational study with the different algorithms cited above. The paper is organized as follows: section 2 describes the problem, in section 3 we propose an heuristic algorithm and analyze its computational complexity. In section 4 the computational experience is reported. Finally, section 5 presents conclusions and final remarks.

2. The problem

A multifunctional machine has to perform a set of n tasks or jobs $\{J_1,...,J_n\}$. Each job j has a known processing time p_j and a given due date d_j. The jobs must be performed without preemption, i.e., when the

machine starts with the processing of a job *j*, the machine has to continue the processing without any pause. The jobs must be available to be processed at time zero; furthermore, there are no precedence relations among them.

When the multifunctional machine realizes one of its functions, or when the machine performs a type of task, the machine has to follow a sequence of instructions. These instructions will come from a control device. We assume that the times used by the control device to interchange one set of instructions for another are negligible. Note that we are not talking about set-up times because the multifunctional machine is always available to perform any of its functions. We accept that these instruction set interchange times are insignificant in the model because the control devices are usually computerized and the selection among different sets of instructions is, in practice, an instantaneous selection.

Let C_j be the completion time of the job *j*, i.e., the instant when this job is finished. This instant varies in relation to the position that the job *j* occupies in the job sequence. If job *j* occupies the position *k*, then $C_j = \sum_{i=1}^{k} p_{[i]}$, where *[i]* denotes the job at position *i*.

The value of the tardiness of job *j* is $T_j = max\{0, L_j = C_j - d_j\}$. The total tardiness problem, denoted by $1||\sum T_j$, consists of finding a sequence $s = [[1],[2],...,[n]]$ of jobs in such a way the total tardiness $\sum_{j=1}^{n} T_{[j]}$ is minimized. Note that after the tasks are performed by the machine, the processing continues in others machines. Consequently, it is important that such tasks be completed before their due dates.

Now, we propose an heuristic method to solve the problem.

3. Algorithm

The proposed algorithm can be described as follows. An initial feasible solution is chosen to be the current solution. Then, the neighbor sequences are generated by changing the position of a previously selected job in the sequence. Among the generated neighbors, we select the sequence with the least total tardiness. If this sequence is better than the current sequence, the current sequence is updated, new neighbor sequences are again generated, and another iteration in the search is made. The algorithm stops when the current sequence is better than all its neighbors.

To select the initial feasible sequence, it is useful to keep in mind different results from specific instances of the problem. In this sense, Baker (1974) points out that, when the earliest due date sequence (*EDD* sequence) has, at most, one tardy job $(T_j > 0)$, then this sequence is optimal. Also, he affirms that if all the jobs have the same due date, that is, $d_j = d$, $"j\ 1= 1, ...,n$, then the shortest processing time sequence (*SPT* sequence) is an optimal one.

However, in more general cases of the problem, it is not easy to define optimal sequences.

When the neighbor sequences are generated, we can compute their total tardiness without again computing all the completion times of the jobs. These computations are done using the theorem proposed below. To describe the theorem we need the following notation:

- *[j]* = job in position *j*
- $[j_o]$ = candidate job to be changed from position j_o to another one
- $B_{j_o\text{-}m}$ = decreasing value of the completion time of job $[j_o]$ when job $[j_o]$ is moved earlier from position j_o to position j_o-*m*.
- A^m = increasing the tardiness by postponing the jobs $[j_o\text{-}m], [j_o\text{-}m+1], ..., [j_o\text{-}1]$.
- D_a^m = difference between the total tardiness of the current solution and the total tardiness from the sequence obtained when the job $[j_o]$ is moved to the left to the position j_o-*m* (anticipation). This difference could be negative.
- DC_m = increasing of the completion time of job $[j_o]$ when it is postponed from position j_o to position j_o+*m*.
- Y^m = increasing of the tardiness of job $[j_o]$ when it is postponed from position j_o to position j_o+*m*.
- Z^m = decreasing of the tardiness due to the fact of anticipating the jobs $[j_o+1], ..., [j_o+m]$.
- D_b^m = difference between the total tardiness of the current sequence and the total tardiness of the sequence obtained when the job $[j_o]$ is moved to the right to the position j_o+*m* (delay). This difference could be negative.

Theorem

Let j_o be the position of the candidate job $[j_o]$ in the current sequence. Let *T* be the total tardiness in this sequence. Then, the following statements are true:

a) if the candidate job is moved to the position j_o-*m*, the new sequence has a total tardiness $T' = T - D_a^m$, where D_a^m is recursively computed by the formulas:

$$\Delta T_{j_o-m}=\begin{cases} p_{[j_o]}, & if \quad L_{[j_o-m]}>0 \\ 0, & if \quad L_{[j_o-m]}\le 0 \text{ and } p_{[j_o]}\le \left|L_{[j_o-m]}\right| \\ L_{[j_o-m]}+p_{[j_o]}, & if \quad L_{[j_o-m]}\le 0 \text{ and } p_{[j_o]}> \left|L_{[j_o-m]}\right| \end{cases}$$

$$A^0 = 0;\ A^m = A^{m-1} + DT_{j_O-m};$$

$$B_{j_O} = 0,\ B_{j_O-m} = B_{j_O-m+1} + p_{[j_O-m]}$$

$$D_a{}^m = min\{B_{j_O-m}, T_{[j_O]}\} - A^m.$$

b) if the candidate job is moved to position j_O+m, the new sequence has a total tardiness $T' = T - D_b{}^m$, where $D_b{}^m$ is recursively computed by the formulas:

$$DC_O = 0;\ DC_m = DC_{m-1} + p_{[j_O+m]}$$

$$Y^m=\begin{cases} \Delta C_m, & if \quad L_{[j_o]}>0 \\ max\left\{0, L_{[j_o]}+\Delta C_m\right\}, & if \quad L_{[j_o]}\le 0 \end{cases}$$

$$X_{j_o+m}=\begin{cases} 0, & if \quad L_{[j_o+m]}\le 0 \\ min\left\{L_{[j_o+m]}, p_{[j_o]}\right\}, & if \quad L_{[j_o+m]}>0 \end{cases}$$

$$Z^O = 0,\ Z^m = Z^{m-1} + X_{j_O+m};$$

$$D_b{}^m = Z^m - Y^m.$$

Proof

a) If the candidate job $[j_O]$ is moved to the left, only the tardiness $T_{[j]}$ with $j = j_O-m, j_O-m+1, ..., j_O$ will change (see figure 1) according to the formulas:

$$T'_{[j_o]}=\begin{cases} T_{[j_o]}-\sum_{j=j_o-m}^{j_o-1} p_{[j]}, & if \ \sum_{j=j_o-m}^{j_o-1} p_{[j]}\le T_{[j_o]} \\ 0, & otherwise \end{cases}$$

$$T'_{[j]} = max\left\{0, L_{[j]}+p_{[j_o]}\right\},\ j = j_o-m,..., j_o-1.$$

Then, the tardiness $T'[j_O]$ is reduced in the value:

$$min\left\{\sum_{j=j_o-m}^{j_o-1} p_{[j]}, T_{[j_o]}\right\} = min\left\{B_{j_o-m}, T_{[j_o]}\right\}$$

when the job $[j_O]$ is moved to position j_O-m. However, the tardiness $T_{[j_O-m+i]}$ with $i = 0, 1, ..., m-1$, increases due to the increasing of completion times.

$$\Delta T_{j_o-m+i}=\begin{cases} p_{[j_o]}, & if \quad T_{[j_o-m+i]}>0 \\ 0, & if \quad T_{[j_o-m+i]}=0 \text{ and } p_{[j_o]}\le \left|L_{[j_o-m+i]}\right| \\ L_{[j_o-m+i]}+p_{[j_o]}, & if \quad T_{[j_o-m+i]}=0 \text{ and } p_{[j_o]}> \left|L_{[j_o-m+i]}\right| \end{cases}$$

Then, $A^m = \sum_{i=0}^{m-1} C_{j_o-m+i}$ represents the generated increment.

Hence, $D_a^m = min\left\{B_{j_o-m}, T_{[j_o]}\right\} - A^m$ is the total advantage in the change.

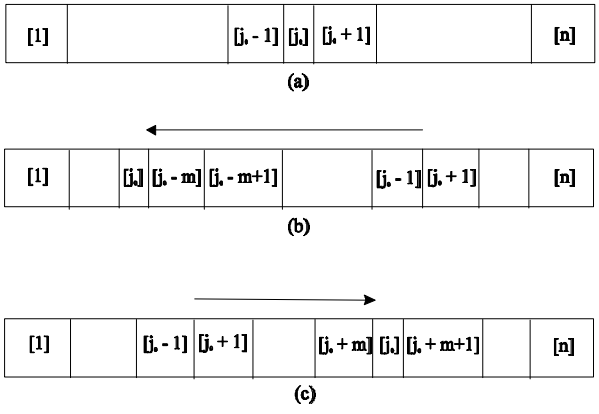

Figure 1. (a) Current sequence, (b) neighbor: the candidate job is moved to the left, (c) neighbor: the candidate job is moved to the right.

b) In the case in which the candidate job is moved to the right the proof is similar. So, only the tardiness $[j_O]$ with $j = j_O, j_O+1, ..., j_O+m$ could change (see figure 1) according to the formulas:

$$T'_{[j_o]} = max\left\{T_{[j_o]}, L_{[j_o]}+\sum_{i=1}^{m} p_{[j_o+i]}\right\} = max\left\{T_{[j_o]}, L_{[j_o]}+F_m\right\}$$

and for $j = j_O+1, ..., j_O+m$ the new tardiness is:

$$T'_{[j]}=\begin{cases} 0, & if \ L_{[j]}\le 0 \\ max\left\{0, L_{[j]}-p_{[j_o]}\right\}, & if \ L_{[j]}>0 \end{cases}$$

Then the tardiness $T_{[j_O]}$ increases by

$$max\left\{0, -T_{[j_o]}+L_{[j_o]}+\sum_{i=1}^{m} p_{[j_o+i]}\right\} = Y^m$$

while for each $j = j_O+1,...,j_O+m$ the tardiness $T_{[j]}$ decreases by $min\{L_{[j]}, p_{[j_O]}\}$ if $L_{[j]} > 0$, and remains the same in the remaining cases.

Hence, the obtained reduction is $Z^m = \sum_{j \in J'} min\{L_{[j]}, p_{[j_O]}\}$, where $J'=\{j / j_O+1 \le j \le j_O+m \text{ and } L_{[j]} > 0\}$, and the total advantage in this change will be $D_b{}^m = Z^m - Y^m$.

As the *EDD* sequence minimizes the maximum tardiness, we consider this sequence as the initial or seed sequence. If in the sequence there are two or more jobs with the same due dates, we break the tie by sorting these jobs according to no decreasing processing times.

We propose an heuristic method for the total tardiness problem based on the above theorem. The algorithm works in the following way: first, the candidate job to be moved is chosen (step 2); next, the better change is determined, i. e., if we have to anticipate or to delay the selected job. The variation of the total tardiness value if we anticipate the job is computed in steps 4 and 5. Steps 7 and 8 compute the variation if we delay the candidate job. The decision to move the job to the best position is considered in step 9.

If we do not make any change, then the total tardiness is not reduced using this candidate job and we select another candidate job. The process stops when there is not any change that improves the total tardiness.

The algorithm could be presented in the following way:

Algorithm

1. Determine the EDD sequence, which will be seed or initial sequence. Let *[[1],[2],...,[n]]* be the job sequence. Compute $L_{[j]} = \sum_{i=1}^{j} p_{[i]} - d_{[j]}$ and $T_j = max\{0, L_{[j]}\}$ "j = 1,...,n. Calculate $T = \sum_{j=1}^{n} T_{[j]}$. Initialize the list of tested jobs, i.e., $P = \AE$.
2. Choose the index j_O such that $L_{[j_O]} = max\{L_{[j]} / 1 \le j \le n\}$. The job $[j_O]$ is the candidate to change position.
3. If $j_O = 1$, go to step 6. If $j_O \neq 1$, but $T_{[j_O]} = 0$, go to step 6. Otherwise, set $m=1$, $G = 0$, $q = 0$, $A^0 = 0$, $B_{j_O} = 0$.
4. *(Anticipation)*. If $m=j_O$, go to step 6. Otherwise, compute the following values:

$$\Delta T_{j_o-m} = \begin{cases} p_{[j_o]}, & if \quad L_{[j_o-m]} > 0 \\ 0, & if \quad L_{[j_o-m]} \le 0 \text{ and } p_{[j_o]} \le \left| L_{[j_o-m]} \right| \\ L_{[j_o-m]} + p_{[j_o]}, & if \quad L_{[j_o-m]} \le 0 \text{ and } p_{[j_o]} > \left| L_{[j_o-m]} \right| \end{cases}$$

$$A^m = A^{m-1} + DT_{j_O-m} \; ; \; B_{j_O-m} = B_{j_O-m+1} + p_{[j_O-m]}$$

$$D^m = min\{B_{j_O-m}, T_{[j_O]}\} - A^m.$$

5. - If $D^m = T_{[j_O]}$, then set $q = j_O - m$, $G = D^m$, and go to step 6.
   - If $D^m \neq T_{[j_O]}$ and $D^m > G$, then set $q = j_O - m$, $G = D^m$. Put $m=m+1$ and go to step 4.
   - If $D^m \neq T_{[j_O]}$ and $D^m \le G$, then put $m=m+1$ and go to step 4.
6. If $j_O = n$, go to step 9. If $j_O \neq n$, set $m=1, DC_O=0$, and $Z^O = 0$.
7. *(Delay)*. If $m = n - j_O + 1$, go to step 9. Otherwise, compute the following values:

$$DC_m = DC_{m-1} + p_{[j_O+m]}$$

$$Y^m = \begin{cases} \Delta C_m, & if \quad L_{[j_o]} > 0 \\ max\{0, L_{[j_o]} + \Delta C_m\}, & if \quad L_{[j_o]} \le 0 \end{cases}$$

$$X_{j_o+m} = \begin{cases} 0, & if \quad L_{[j_o+m]} \le 0 \\ min\{L_{[j_o+m]}, p_{[j_o]}\}, & if \quad L_{[j_o+m]} > 0 \end{cases}$$

$$Z^m = Z^{m-1} + X_{j_O+m} \; ; \; D^m = Z^m - Y^m.$$

8. - If $D^m > G$, set $q = j_O + m$ and $G = D^m$. Set $m = m + 1$ and go to step 7.
   - If $D^m \le G$, set $m = m + 1$ and go to step 7.
9. *(Change)*. If $q = 0$, no changes are made. Set $P = P \cup \{j_O\}$ and go to step 10 (searching for a new candidate).
   - If $q > 0$, move the job $[j_O]$ to the position q and *"slide"* all the necessary jobs. So, let $MIN = min\{q, j_O\}$, and $MAX = max\{q, j_O\}$.
   - if $MIN = q$, set $s(q) = [j_O]$, $s(q+1) = [q]$,..., $s(q+k) = [q+k-1]$,..., $s(j_O) = [j_O-1]$.
   - if $MIN = j_O$, set $s(j_O) = [j_O+1]$, ..., $s(j_O+k) = [j_O+k+1]$, ..., $s(q-1) = [q]$, $s(q) = [j_O]$.

   Before finishing this step, rearrange the sequence in such a way that the job at position i of the sequence is $[i] = s(i)$, $" i \in [MIN, MAX]$.

   Compute again $L_{[j]}$ and $T_{[j]}$ for all $j \in [MIN, MAX]$ using the formulas of step 1. Set $T = T - G$, $P = \AE$ and go to step 2.
10. - If $card(P) = n$, stop. The sequence *[[1],[2],...,[n]]* is the solution given by the algorithm, being T the value of the total tardiness for this sequence.
    - If $card(P) \neq n$, then find the index j_1 such that $L_{[j_1]} = max\{L_{[k]} / k \notin P\}$. Set $j_O=j_1$ and go to step 3.

Now, we study the computational complexity of this algorithm. The algorithm has a computational complexity of $O(n^2 \sum_{j=1}^{n} p_j)$. This amount is computed in the following way. Step 1 requires $O(n \log n)$ operations, while steps 2 to 10 require $O(n)$ for each iteration. The number of iterations for these steps is at most $\sum_j T_j(EDD) - T_{max}(EDD)$, being $T_j(EDD)$ the tardiness of job j in the *EDD* sequence and $T_{max}(EDD)$ the maximum tardiness in this sequence. The key is to determine the value of this expression. The interval of variation of this value is bounded by $n \sum_{j=1}^{n} p_j$. Hence, the recurrence of steps 2 to 10 gives us a complexity of $0(n^2 \sum_{j=1}^{n} p_j)$ which is greater than the complexity of step 1 and, as a consequence of that, this expression is the complexity of the algorithm.

Note that the value $\sum_j T_j(EDD) - T_{max}(EDD)$ is computed in step 1 and at that time, we can predict the heuristic behavior with respect to the convergence speed of the algorithm. It is not necessary to run the entire algorithm for it.

In the next section we report a computational study where the proposed algorithm and algorithms of other authors have also been codified and tested.

4. Computational Experience

In this section we do a comparative computational study among exact algorithms, the proposed heuristic algorithm and other heuristic algorithms. All these algorithms are codified in C programming language. We use a personal compatible computer 80486 DX with 50 Mhz.

The test problems used are the problems generated by Potts and van Wassenhove (1982). With this generator, the processing times of each job are taken from *U[1,100]*. Next, the due dates *dj* are generated. For that, we compute the amount of the processing times $P = \sum pj$ and take the due dates from the $U\left[P(1 - TF - \frac{RDD}{2}),\ P(1 - TF + \frac{RDD}{2}) \right]$ where the parameters *TF* and *RDD* vary in the set of values {0.2, 0.4, 0.6, 0.8, 1.0} and {0.2, 0.4, 0.6, 0.8} respectively. Hence, for each value of *n*, and each set of processing times generated we build 20 instances. We have fixed a running time limit of 5 minutes for the exact algorithms.

The considered *dynamic programming exact algorithms* are: basic dynamic programming algorithm of Baker, dynamic programming algorithm of Srinivasian, dynamic programming algorithm of Lawler, dynamic programming algorithm of Lawler with the dominance properties of Elmaghraby, dynamic programming algorithm of Schrage and Baker, and the dynamic programming algorithm Schrage and Baker with the dominance properties of Elmaghraby. Computational experiences of these algorithms are reported in tables 1 and 2.

Also *exact algorithms based on decomposition theorems* have been considered. These algorithms are: decomposition algorithm of Lawler, decomposition algorithm of Lawler with reduced search, and the decomposition algorithm of Potts and van Wassenhove. The computational study of these algorithms is reported in the third table.

The *heuristic algorithms* considered are: algorithm of Wilkerson and Irwin, algorithm of Fry, Macleod, Vicens and Fernández, algorithm of Holsenback and Russell, and, finally, the proposed heuristic algorithm. Computational results are given in tables 4 and 5.

The times are CPU times of a personal computer 80486 DX with 50 Mhz.

The column titles in tables 1, 2 and 3 mean the following:

n = number of jobs

nprob = number of generated problems

solved = number of solved problems before the time limit.

t_average = average running times.

t_worst = maximum running time.

sets = number of sets generated with the dynamic programming formulas.

Basic DP Baker					
n	***nprob***	***solved***	***t_average***	***t_worst***	***sets***
5	*100*	*100*	*0.007*	*0.055*	*32*
10	*100*	*100*	*0.341*	*0.549*	*1024*
15	*100*	*100*	*30.212*	*30.879*	*32768*
DP Srinivasian					
n	***nprob***	***Solved***	***t_average***	***t_worst***	***sets***
5	*100*	*100*	*0.001*	*0.055*	*5*
10	*100*	*100*	*0.026*	*0.330*	*79*
15	*100*	*100*	*2.969*	*30.440*	*3796*
DP Lawler					
n	***nprob***	***Solved***	***t_average***	***T_worst***	***sets***
5	*100*	*100*	*0.015*	*0.055*	*6*
10	*100*	*100*	*0.025*	*0.055*	*18*
15	*100*	*100*	*0.044*	*0.330*	*53*
20	*100*	*100*	*0.120*	*1.209*	*111*
30	*100*	*100*	*1.510*	*46.978*	*763*
40	*100*	*98*	*16.829*	*202.033*	*2997*
50	*100*	*88*	*24.435*	*283.462*	*3649*
60	*100*	*79*	*12.792*	*155.769*	*2724*
70	*100*	*59*	*4.395*	*32.143*	*1083*
80	*100*	*58*	*4.308*	*64.121*	*954*
90	*100*	*57*	*6.356*	*43.846*	*1287*
DP Schrage and Baker					
n	***nprob***	***solved***	***t_average***	***T_worst***	***sets***
5	*100*	*100*	*0.003*	*0.055*	*6*
10	*100*	*100*	*0.013*	*0.110*	*18*
15	*100*	*100*	*0.083*	*1.429*	*55*
20	*100*	*100*	*0.338*	*7.473*	*115*
30	*100*	*100*	*34.151*	*109.967*	*808*
40	*100*	*79*	*14.197*	*205.604*	*467*
50	*100*	*69*	*20.091*	*243.462*	*581*
60	*100*	*61*	*33.034*	*245.000*	*742*
70	*100*	*56*	*14.947*	*169.505*	*496*
80	*100*	*56*	*21.580*	*227.143*	*493*
90	*100*	*51*	*18.234*	*254.231*	*426*

Table 1. Computational results for exact algorithms of dynamic programming (DP): basic DP of Baker, DP of Srinivasian, DP of Lawler, DP of Schrage and Baker.

DP Lawler (with Elmaghraby)					
n	***nprob***	***solved***	***t_average***	***T_worst***	***sets***
5	*100*	*100*	*0.016*	*0.055*	*6*
10	*100*	*100*	*0.018*	*0.055*	*16*
15	*100*	*100*	*0.039*	*0.275*	*44*
20	*100*	*100*	*0.105*	*0.989*	*84*
30	*100*	*100*	*0.737*	*11.099*	*428*
40	*100*	*100*	*11.485*	*189.121*	*1968*
50	*100*	*93*	*17.919*	*182.637*	*2731*
60	*100*	*85*	*15.608*	*271.099*	*2418*
70	*100*	*70*	*9.992*	*84.945*	*1719*
80	*100*	*66*	*7.959*	*79.890*	*1235*
90	*100*	*69*	*11.222*	*94.560*	*1606*
DP Schrage and Baker (with Elmaghraby)					
n	***nprob***	***Solved***	***t_average***	***t_worst***	***sets***
5	*100*	*100*	*0.004*	*0.055*	*6*
10	*100*	*100*	*0.013*	*0.165*	*18*
15	*100*	*100*	*0.079*	*1.429*	*55*
20	*100*	*100*	*0.334*	*7.418*	*115*
30	*100*	*100*	*34.171*	*109.626*	*808*
40	*100*	*79*	*14.220*	*205.714*	*467*
50	*100*	*69*	*20.119*	*243.516*	*581*
60	*100*	*61*	*33.034*	*245.440*	*742*
70	*100*	*56*	*14.981*	*170.000*	*496*
80	*100*	*56*	*21.635*	*227.198*	*493*
90	*100*	*51*	*18.295*	*254.231*	*426*

Table 2. Computational results for exact dynamic programming algorithms DP of Lawler, and DP of Schrage and Baker using dominance rules done by Elmaghraby.

Decomposition Algorithm of Lawler					
n	***nprob***	***solved***	***t_average***	***t_worst***	***Sets***
5	*100*	*100*	*0.002*	*0.055*	*12*
10	*100*	*100*	*0.116*	*0.330*	*498*
15	*100*	*100*	*4.640*	*16.648*	*18639*
Dec. Alg. of Lawler (with reduced search)					
n	***nprob***	***Solved***	***t_average***	***t_worst***	***sets***
5	*100*	*100*	*0.003*	*0.055*	*7*
10	*100*	*100*	*0.073*	*0.275*	*236*
15	*100*	*100*	*3.108*	*22.747*	*9254*

Potts and Van Wassenhove					
n	***nprob***	***solved***	***t_average***	***t_worst***	***Sets***
5	*100*	*100*	*0.001*	*0.055*	*4*
10	*100*	*100*	*0.003*	*0.055*	*11*
15	*100*	*100*	*0.004*	*0.055*	*25*
20	*100*	*100*	*0.015*	*0.055*	*45*
30	*100*	*100*	*0.054*	*0.440*	*187*
40	*100*	*100*	*0.130*	*0.824*	*411*
50	*100*	*100*	*0.383*	*6.923*	*1179*
60	*100*	*100*	*0.861*	*24.505*	*2669*
70	*100*	*100*	*1.938*	*27.582*	*6012*
80	*100*	*100*	*4.751*	*155.879*	*14373*
90	*100*	*100*	*4.755*	*71.374*	*14420*
100	*100*	*100*	*10.386*	*225.110*	*29765*
110	*100*	*100*	*25.332*	*380.495*	*68583*
120	*100*	*100*	*104.676*	*4558.846*	*274545*
130	*100*	*100*	*90.390*	*1572.912*	*230954*
140	*100*	*100*	*413.945*	*17608.297*	*1038676*
150	*100*	*100*	*591.719*	*22586.209*	*1503148*

Table 3. Computational results for exact decomposition algorithms of Lawler, and Potts and van Wassenhove.

Tables 4 and 5 refer to heuristic algorithms. In these cases there is no fixed time limit. The heuristic algorithms have been compared with the exact algorithm of Potts and van Wassenhove. We compute the ratio: optimal solution / heuristic solution. The titles of the columns have the following meaning:

n = number of jobs

nprob = number of generated problems

t_exact = average running time of exact algorithm.

t_heur = average running time of the heuristic algorithm

% = average ratio $\frac{z^*}{z_h} x100$, where z^* is the value of the optimal solution and z_h the value of the heuristic solution.

Wilkerson and Irwin				
n	***nprob***	***%***	***t_exact***	***t_heur***
10	*100*	*98.10*	*0.004*	*0.000*
20	*100*	*98.68*	*0.015*	*0.002*
30	*100*	*96.39*	*0.051*	*0.002*
40	*100*	*96.58*	*0.115*	*0.004*
50	*100*	*95.75*	*0.336*	*0.008*
60	*100*	*95.03*	*0.788*	*0.015*
70	*100*	*94.97*	*2.027*	*0.024*
80	*100*	*95.24*	*5.004*	*0.034*
90	*100*	*96.39*	*4.992*	*0.045*
100	*100*	*95.47*	*10.960*	*0.063*
110	*100*	*95.80*	*25.332*	*0.102*
120	*100*	*95.41*	*104.676*	*0.147*
130	*100*	*95.94*	*90.390*	*0.206*
140	*100*	*94.80*	*413.945*	*0.251*
150	*100*	*93.89*	*591.719*	*0.322*
Fry, Macleod, Vicens and Fernández				
N	***nprob***	***%***	***t_exact***	***t_heur***
10	*100*	*99.58*	*0.004*	*0.003*
20	*100*	*99.30*	*0.015*	*0.013*
30	*100*	*98.41*	*0.051*	*0.035*
40	*100*	*98.41*	*0.115*	*0.071*
50	*100*	*98.41*	*0.336*	*0.134*
60	*100*	*98.23*	*0.788*	*0.223*
70	*100*	*98.51*	*2.027*	*0.349*
80	*100*	*98.52*	*5.004*	*0.521*
90	*100*	*98.48*	*4.992*	*0.722*
100	*100*	*98.42*	*10.960*	*0.981*

110	*100*	*98.49*	*25.332*	*1.304*
120	*100*	*98.05*	*104.676*	*1.668*
130	*100*	*98.20*	*90.390*	*2.116*
140	*100*	*98.36*	*413.945*	*2.624*
150	*100*	*98.22*	*591.719*	*3.184*

Table 4. Computational results for heuristic algorithms of Wilkerson and Irwin, and Fry, Macleod, Vicens and Fernández.

Holsenback and Russell				
n	***nprob***	***%***	***T_exact***	***t_heur***
10	*100*	*99.92*	*0.004*	*0.002*
20	*100*	*99.71*	*0.015*	*0.003*
30	*100*	*99.35*	*0.051*	*0.004*
40	*100*	*99.08*	*0.115*	*0.007*
50	*100*	*98.92*	*0.336*	*0.008*
60	*100*	*98.50*	*0.788*	*0.012*
70	*100*	*98.21*	*2.027*	*0.015*
80	*100*	*98.56*	*5.004*	*0.022*
90	*100*	*98.77*	*4.992*	*0.027*
100	*100*	*97.96*	*10.960*	*0.027*
110	*100*	*98.52*	*25.332*	*0.034*
120	*100*	*98.51*	*104.676*	*0.042*
130	*100*	*98.20*	*90.390*	*0.044*
140	*100*	*97.68*	*413.945*	*0.057*
150	*100*	*97.85*	*591.719*	*0.060*
Proposed heuristic algorithm				
n	***nprob***	***%***	***t_exact***	***t_heur***
10	*100*	*97.75*	*0.005*	*0.002*
20	*100*	*97.57*	*0.015*	*0.014*
30	*100*	*97.20*	*0.051*	*0.045*
40	*100*	*97.98*	*0.114*	*0.108*
50	*100*	*97.97*	*0.342*	*0.236*
60	*100*	*98.12*	*0.785*	*0.432*
70	*100*	*97.26*	*2.027*	*0.754*
80	*100*	*98.37*	*5.004*	*1.168*
90	*100*	*97.98*	*4.992*	*1.761*
100	*100*	*98.42*	*10.960*	*2.593*
110	*100*	*98.75*	*25.332*	*3.680*
120	*100*	*98.52*	*104.676*	*4.954*
130	*100*	*98.14*	*90.390*	*6.521*
140	*100*	*98.24*	*413.945*	*8.423*
150	*100*	*98.25*	*591.719*	*10.774*

Table 5. Computational results for heuristic algorithm of Holsenback and Russell, and the proposed heuristic algorithm.

From computational experience we observe that the first dynamic programming algorithms (basic dynamic programming algorithm of Baker and Srinivasian's algorithm) are limited since the number of sets to explore with dynamic programming increase exponentially when the number of jobs are increased. Later, dynamic programming algorithms like the algorithm of Lawler and the algorithm of Schrage and Baker, avoid to a certain extent this difficulty and allow us to solve medium-sized problems. Applying in those algorithms dominance rules like Elmaghraby's rules, it is possible sometimes to reduce the number of sets to explore and the total memory requirements.

Using the decomposition algorithms, the reduction is more significant. As the memory requirements are smaller, it allows us to solve problems with more jobs. This is specially clear with Potts and van Wassenhove's algorithm. This algorithm is the exact algorithm with the best behavior in the test problems considered. Hence, we have taken this exact algorithm as the reference to study the heuristic algorithms.

Taking into account the complexity of the problem, and as the computational experience shows, the heuristic algorithms seem to be the best way to achieve acceptable solutions in reasonable times when the number of jobs is 100 or more. Tables 4 and 5 show that the heuristics guaranteed a precision of approximated 98%. If we observe the precision of heuristic algorithms, when the number of jobs is less than 100, the best one is the Holsenback and Russell heuristic, next the Fry, Vicens, Macleod and Fernández one, the proposed heuristic, and finally the Wilkerson and Irwin one. However, if the number of jobs is greater than 100, in precision terms, the best heuristic seems to be the proposed heuristic, next the Fry, Vicens, Macleod and Fernández one, the Holsenback and Russell heuristic, and the Wilkerson and Irwin one.

Note that for n greater than 100, the precision of the proposed heuristic and the heuristic of Fry, Vicens, Macleod and Fernández tend to increase when n increases and always with a precision greater than 98%; while the precision of the Wilkerson and Irwin's heuristic and Holsenback and Russell's heuristic tend to decrease and are less than 98% when n increases.

5. Conclusions and final remarks

Multifunctional machines are commonly used in industrial process. These machines can be considered as executing intermediate steps in the process. The output tasks of some machines are input tasks for other machines. To complete certain industrial processes, we need to perform all the tasks involved in the process. To do it with optimality, it can be useful to schedule advantageously the different tasks of each multifunctional machine.

In this paper we have considered the problem of scheduling the activities of a multifunctional machine in such a way that the machine complies with the due dates of its tasks. The problem is formulated as a minimum total tardiness scheduling problem. An

heuristic algorithm to solve the problem is proposed and a comparative computational study of this heuristic with other heuristic and exact algorithms is reported.

Future developments of this work could be to situations in which task preemptions are allowed, there are precedence relations among tasks, or the tasks have different release dates.

References

[1]. D. Alcaide, *Problemas de planificación y secuenciación determinística: modelización y técnicas de resolución,* Ph.D. Thesis. DEIOC. Universidad de La Laguna, 1995.

[2]. K.R. Baker, *Introduction to sequencing and scheduling,* John Wiley, 1974.

[3]. J. Blazewicz, Selected topics in scheduling theory, *Annals of Discrete Mathematics,* Vol. 31, 1987, pp. 1-60.

[4]. P. Chrétienne, E.G. Coffman, Jr., J.K. Lenstra and Z. Liu (eds.) *Scheduling theory and its applications,* John Wiley and Sons, 1995.

[5]. J. Du and J.Y.-T. Leung, Minimizing total tardiness on one machine is NP-hard, *Math. of Oper. Res.* Vol. 15, No. 3, 1990, pp. 483-495.

[6]. H. Emmons, One machine sequencing to minimize certain functions of job tardiness, *Oper. Res.* Vol. 17, 1969, pp. 701-715.

[7]. T.D. Fry, L. Vicens, K. Macleod and S. Fernández, A heuristic solution procedure to minimize $\overline{T}$ on a single machine, *Journal of the Operations Research Society*, Vol. 40, No. 3, 1989, pp. 293-297.

[8]. J.E. Holsenback and R.M. Russell, A heuristic algorithm for sequencing on one machine to minimize total tardiness, *Journal of the Operations Research Society*, Vol. 43, No. 1, 1992, pp. 53-62.

[9]. E.L. Lawler, A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness, *Annals of Discrete Mathematics*, Vol. 1, 1977, pp. 331-342.

[10]. E.L. Lawler, Efficient implementation of dynamic programming algorithms for sequencing problems, *Report BW 106, Mathematisch Centrum*, Amsterdam, 1979.

[11]. J.K. Lenstra and A.H.G. Rinnooy Kan, Complexity results for scheduling chains on a single machine, *European Journal of Operations Research*, Vol. 4, 1980, pp. 270-275.

[12]. C.N. Potts and L.N. Van Wassenhove, A decomposition algorithm for the single machine tardiness problem, *Operations Research Letters*, Vol. 1, No. 5, 1982, pp. 177-181.

[13]. C.N. Potts and L.N. Van Wassenhove, Dynamic programming and decomposition approaches for the single machine tardiness problem, *European Journal of Operations Research*, Vol. 32, 1987, pp. 405-414.

[14]. Schrage and K.R. Baker, Dynamic programming solution of sequencing problems with precedence constraints, *Oper. Res.*, Vol. 26, 1978, pp. 444-449.

[15]. V. Srinivasian, A hybrid algorithm for the one machine sequencing problem to minimize total tardiness, *Naval Research Logistic Quarterly*, Vol. 18, No. 3, September, 1971.

[16]. L.J. Wilkerson and J.D. Irwin, An improvement algorithm for scheduling independent tasks, *A.I.I.E. Trans.* Vol. 3, 1971, pp. 239-245.