Journal of Intelligent Manufacturing (1991) **2**, 253–260

A search heuristic for just-in-time scheduling in parallel machines

MANUEL LAGUNA[1] and JOSÉ LUIS GONZÁLEZ VELARDE[2]

[1]*US West Postdoctoral Fellow, Graduate School of Business and Administration, Campus Box 419, University of Colorado at Boulder, Boulder, CO 80309-0419, USA*
[2]*Centro de Sistemas de Manufactura, División de Graduados e Investigación, ITESM, Sucursal de Correos J, Monterrey, NL 64849, México*

Received November 1990 and accepted March 1991

In recent years the Just-in-Time (JIT) production philosophy as been adopted by many companies around the world. This has motivated the study of scheduling models that embrace the essential components of JIT systems. In this paper, we present a search heuristic for the weighted earliness penalty problem with deadlines in parallel identical machines. Our approach combines elements of the solution methods known as greedy randomized adaptive search procedure (GRASP) and tabu search. It also uses a branch-and-bound post-processor to optimize individually the sequence of the jobs assigned to each machine.

Keywords: Tabu search, GRASP, just-in-time, scheduling

1. Introduction

The Just-in-Time (JIT) production system was developed by Toyota Motor Co., Ltd, and has been adopted by many companies around the world. JIT basically means to produce the necessary units in the necessary quantities at the necessary time. The main purpose of JIT systems is to reduce costs associated with production processes thus improving the total productivity of a company. The current interest in JIT production has motivated the study of scheduling models capable of capturing the essence of this idea. Unfortunately, as shown in the review article by Baker and Scudder (1990), very little has been done in terms of developing solution procedures for scheduling problems in JIT environments with more than one machine.

In this paper we study the multi-machine weighted earliness (WE) problem with deadlines. As opposed to due dates, which may be violated at the cost of tardiness, deadlines must be met and cannot be violated. The WE problem is specially relevant to companies with *point-of-use* manufacture. This manufacturing approach means that the work-stations making the components are located along the assembly line immediately before the assembly operations they serve. In this way, point-of-use manufacture increases productivity by reducing the amount of work-in-process. At the same time, failing to meet deadlines causes the entire line to shut down.

We encountered an instance of the WE problem in the last stage of a JIT system for the manufacturing of thread. At this stage, n jobs are to be processed on m machines in order to meet deadlines and minimize the total earliness penalty. Each job, j, is described by a processing time, t_j, a deadline, d_j, and an earliness penalty, p_j. As a result of scheduling decisions, job j will be assigned a position in one of the m machines and a completion time, denoted by c_j.

A schedule, S, has the following form

$$S = \{\Pi, c\}$$

where $\Pi = \{\Pi_1, \Pi_2, \ldots, \Pi_m\}$ is the partition of the n jobs in m sequences, and c is the set of completion times for all jobs. For each machine k, the following represents the sequence in which jobs will be processed

$$\Pi_k = \{\pi_k(1), \pi_k(2), \ldots, \pi_k(n_k), n + k\}$$

where $\pi_k(i)$ is the index of the job in position i on machine

k, and $n+k$ is a dummy job that uniquely identifies the end of each sequence. In mathematical terms, the WE problem can be formulated as follows

$$\text{Minimize} \qquad F(S) = \sum_{j=1}^{n} (d_j - c_j)p_j$$

subject to
$d_j - c_j \geq 0 \qquad \text{for } j = 1, \ldots, n$
and no two jobs overlap.

Since the single machine case of the WE problem is NP-hard (Chand and Schneeberger, 1988) branch-and-bound (Ahmadi and Bagchi, 1986) and dynamic programming (DP) (Chand and Schneeberger, 1988) methods have been developed for its solution. Chand and Schneeberger (1988) also developed a heuristic based on a dispatching rule that selects admissible jobs by their *natural order* (i.e. in increasing value of r_j, where $r_j = t_j/p_j$). The performance of this heuristic was evaluated by comparing its solutions to 10 job problems against optima found with the DP algorithm. Based on extensive computational experiments, the authors were able to identify some weaknesses in their heuristic. For example, problems with tightly clustered deadlines do not have good heuristic solutions because the method does not consider the effect of scheduling a job prior to fellow candidates. The authors also do not recommend the use of their DP procedures for problems with more than 15 jobs. Finally, they identify a need to develop better heuristics for realistic situations.

The primary purpose of our study is to develop a search heuristic to provide high-quality solutions to the WE problem. Our heuristic combines the ideas of two search procedures: GRASP (Feo and Resende, 1989) and tabu search (TS) (Glover, 1989; 1990 de Werra and Hertz, 1989). The TS framework has emerged as an effective approach for handling complex decision problems, and it stems from the general tenets of intelligent problem solving (Glover and Greenberg, 1989). The approach is considered to be a blend of artificial intelligence and operations research, because it employs memory functions to provide an interplay between learning and unlearning that monitors and directs the search procedure.

In Section 2 we present the procedure used to construct initial trial solutions. These solutions are then used to start the TS method presented in Section 3. A branch-and-bound procedure is introduced in Section 4. This procedure is employed as a post-processor optimally to schedule jobs already assigned to each machine. Computational experiments, designed to measure the merit of our solution approach, are shown in Section 5. Conclusions, final remarks, and directions for future research are given in Section 6.

2. GRASP construction

A greedy randomized adaptive search procedure (GRASP) is a technique developed by Feo and Resende (1989) for the approximate solution of combinatorially difficult problems. GRASP typically consists of two phases: the construction step and the local search procedure. The construction phase is based on the idea that a variety of good solutions can be generated by an *intelligent randomization* of the selection step of a greedy heuristic. These solutions are then passed to an exchange procedure that searches for local improvements.

The greedy heuristic we examine for our application is the modified Smith-heuristic (MSH) proposed by Chand and Schneeberger (1988). We first adapt MSH to handle more than one machine and then we integrate a random component. In general terms MSH, as conceived by Chand and Schneeberger (1988), schedules the admissible job with the smallest time-to-penalty ratio (r ratio), starting from the last position. A job is admissible if it belongs to the set of jobs yet to be scheduled (YS), and its deadline is greater than or equal to a variable T. This variable marks the completion time that will be assigned to the selected job.

The deterministic version of MSH for parallel machines is shown in Fig. 1. The first six lines correspond to the initialization step. The set of jobs yet to be scheduled, YS, initially contains all jobs. The planning horizon, h, is defined as the maximum deadline. Each machine is assigned a dummy job and a T value equal to the planning horizon (since machines are filled starting from the last

```
1    $YS \leftarrow [1, \ldots, n]$
2    $h \leftarrow \max_{i \in YS} (d_i)$
3    for $(k = 1, \ldots, m)$ {
4        $T_k \leftarrow h$
5        $\Pi_k \leftarrow \{n+k\}$
6    }
7    do {
8        Find $q$ such that $T_q = \max_{k=1,\ldots,m} (T_k)$
9        Find $j$ such that $r_j = \min_{i \in YS, d_i \geq T_q} (r_i)$
10       $c_j \leftarrow T_q$
11       $\Pi_q \leftarrow \Pi_q \cup \{j\}$
12       $YS \leftarrow YS - \{j\}$
13       for $(k = 1, \ldots, m)$ {
14           $T_k \leftarrow \max_{i \in YS} (d_i)$
15           if $(k = q)$ $T_k \leftarrow \min(c_j - t_j, T_k)$
16       }
17   } while $(YS \neq \emptyset)$
```

Fig. 1. MSH for parallel machines

position, the only admissible jobs are initially those with deadlines exactly equal to h). Lines 7–17 contain the iterative process by which one job is scheduled at a time. In line 8, the index of the machine with the minimum load is found and is labeled machine q. The next job to be scheduled, job j, is the one with the minimum r ratio among all those found admissible (admissibility here depends on the selection of machine q). Line 10 assigns the T value of machine q as the completion time of the selected job j. Line 11 adds job j to the sequence of jobs in machine q. It is important to note that job j is always being inserted in the first position of the given sequence. Line 12 reflects the fact that job j has already been scheduled. The updating of the T values for each machine is carried out in lines 13–16.

A randomized version of the heuristic described above (a procedure that will be referred to as MSH-random), may be achieved by modifying the job selection step of line 9. In MSH-random job j is randomly selected from a candidate list, CL, of admissible jobs (i.e. $i \mid i \in \text{YS}$ and $d_i \geq T_q$). The candidate list is created by first defining r_min and r_max as the minimum and maximum r ratio values, respectively, in the set of admissible jobs. An admissible job i is then added to the list if

$$r_i \leq r\_\text{min} + \alpha(r\_\text{max} - r\_\text{min})$$

where $\alpha\,(0 \leq \alpha \leq 1)$ is a parameter that controls the amount of randomization permitted. If α is set to a value of zero, MSH-random becomes the deterministic procedure described in Fig. 1. On the contrary, the candidate list reaches its maximum possible cardinality when α is set to a value of one.

3. Tabu search

The solutions generated by MSH-random are used as starting-points for a TS method. Tabu search uses flexible memory structures to integrate intensification and diversification strategies. In our implementation (a method that will be referred to as TSH), we make use of a traditional fixed size short-term memory function, as opposed to more elaborate schemes for which attributes are allowed to change their memory size individually. In addition, TSH is designed to identify *essential moves* for specific search states (Laguna and Glover, 1990). Essential moves are those whose execution is considered necessary if an improved solution is to be found. We will further elaborate on this concept later in this section.

Given an initial solution TSH seeks an optimal solution to the WE problem by making a succession of swap or insert moves. The swap move $SP(\pi_k(i), \pi_q(j))$ allows jobs in positions i and j of machines k and q, respectively, to exchange positions. The insert move $IN(\pi_k(i), \pi_q(j))$ consists of transferring the job currently in position i of machine k to a position immediately before job $\pi_q(j)$. Every move has an associated *move value* which is commonly defined as the change on the objective function value, i.e. if S and S' are the schedules before and after the move, then

$$\text{move\_value} = \text{F}(S') - \text{F}(S)$$

In this context moves can be either improving, deteriorating, or null depending on whether their move_value is strictly less than, greater than, or equal to zero, respectively.

A list of candidate moves, along with their associated move values, is made available at every step of the search procedure. The construction of the candidate list considers the exclusion of moves that are expected to have large positive move values. In general, these moves occur when a job is being considered for a position in the schedule that results in a decrease on the completion time of a number of jobs. The candidate list is formed by all moves that fall in one of the following categories

(1) A swap of immediate neighboring jobs on the same machine, i.e. $SP(\pi_k(i), \pi_k(i+1))$ for $i = 1, \ldots, n_k - 1$ and $k = 1, \ldots, \text{m}$.

(2) A swap of two jobs on different machines such that their absolute deadline difference is less than or equal to the threshold value $d$max, i.e. all $SP(\pi_k(i), \pi_q(j))$ for $k \neq q$ such that

$$\text{abs}(d_{\pi_k(i)} - d_{\pi_q(j)}) \leq d\text{max}$$

where $$d\text{max} = \max(1-\rho, 0.25)\,(h - \min_{v=1,\ldots,n}(d_v))$$

and $$\rho = \max\left(\frac{i-1}{n_k}, \frac{j-1}{n_q}\right)$$

(3) An insert move $IN(\pi_k(i), \pi_q(j))$ with the same conditions as in (2), and ρ calculated as below

$$\rho = \frac{j-1}{n_q}.$$

The threshold value $d$max is designed to detect and eliminate from consideration unreasonably large deteriorating moves. Those moves are usually related to a large difference between the deadline values of the jobs being exchanged. The maximum deadline difference is the *deadline range* (i.e. the difference between the planning horizon h and the minimum deadline). The value of $d$max is a percentage of the deadline range that depends on the density measure ρ. If ρ has a value of zero, any deadline difference is acceptable (e.g. when a job is being considered for insertion at the beginning of a sequence). Higher density values reflect the fact that more jobs are

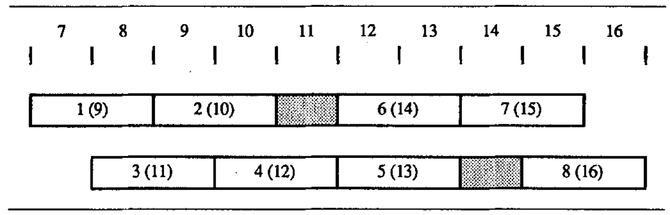

Fig. 2. Schedule for an eight job problem with two machines

scheduled in positions that are earlier than the one occupied by the job under consideration. Swapping or inserting high density positioned jobs with large deadline difference is generally considered unattractive, because many of the jobs in early positions are required to decrease their completion times.

To illustrate this, consider the schedule S of an eight job problem with two machines shown in Fig. 2. All jobs have two units of processing time and unitary earliness penalties. Deadlines are shown between parentheses to the right of each job index. F(S) is currently equal to 5 due to early completion of jobs 1, 3, 4 and 6. Suppose that the swap move $SP(2, 8)$ is under consideration. Its move value is equal to 19. This large move value (almost four times F (S)) is the result of decreasing the completion times of jobs 3, 4, 5 and 8. For this example $d$max has a value of 1.75, and thus it eliminates $S(2, 8)$ from consideration as a candidate move (note that $d_8 - d_2 = 6$). The use of this scheme significantly reduces the computational burden of evaluating a large number of these unattractive moves.

At each iteration TSH selects the *best* candidate move available that is admissible according to the tabu restrictions being imposed. These restrictions are such that after an insert move is executed the transferring job is classified tabu and is not allowed to move during a prescribed number of iterations, *tabu_size*. After a swap move, the tabu job is the one that experienced the largest reduction in its individual earliness penalty (this job is now in a better position than it was prior to the move). A tabu move (one that involves the exchange of any tabu job) may be admissible, provided its execution results in an objective function value that is better than the one of the incumbent solution (i.e. the aspiration level).

The notion of best move is related to the search state (i.e. search history and solution state). The following is the set of conditions that compose the choice rule for the selection of the best move.

(i) If at least one improving move is available, select the most improving move, such that the sum of its move value and the value of the last executed move are different than zero.

(ii) If no improving move that meets condition (ii) is available, select the deteriorating move with the smallest move value from those that exchange the non-tabu job with the largest individual earliness penalty.

Condition (i) states that an improving move is always preferable, provided this move does not cancel the effect that the previously executed move had on the objective function value. This condition, for example, is able to detect and avoid the execution of two consecutive insert moves with a null net effect on the objective function value (i.e. insert moves that are equivalent to a null swap move). Condition (ii) implements the principle that if a better solution will be found later in the search, jobs with large individual penalties must be moved. These moves are classified as *essential*. Due to the diversifying power of essential moves, their execution is particularly important in search states where no admissible move meets condition (i). We have also observed that large improving moves executed early in the search are in fact essential moves.

Calculating a move value, of either an insert or a swap move, requires in the worst case $O(n)$ time. Therefore, given an initial solution the values of all candidate moves are calculated and stored in random access memory (*movalue* array). After a move is executed an updating procedure recalculates only those move values that might have changed as a result of modifications in the current schedule. This strategy was successfully used by Laguna, Barnes and Glover (1990).

Figure 3 shows a *pseudo*-code for the search heuristic, that uses MSH-random constructions and TSH as a local search procedure (this GRASP-TS hybrid will be referred to as GTS). Each solution attempt consists of generating four initial solutions that vary from the most random ($\alpha = 1$) to the deterministic one ($\alpha = 0$). The procedure stores the three solutions below throughout the entire search.

S : The current trial schedule.
S^{α} : The best schedule found for the current α.
S^* : The best overall schedule.

Every time an initial solution is generated the move values for all candidate moves and the tabu structure are initialized (line 6). The tabu structure consists of a single array, *tabu_time*, that records the most recent iteration number at which a job was classified tabu. The updating of the current best solution, S^{α}, and its associated objective function value, F(S^{α}), occurs either after an initial solution is generated (line 4) or after an improving move is executed (line 11). The local search is abandoned if more than 50 moves are executed without improving the objective function value of the best solution found for the current value of α (note that this value may be different than F(S^*) which is the overall best). Line 13 updates the overall best schedule if a better solution was found from the previous starting-point.

An additional criterion not shown in Fig. 3 is used to

```
F(S*) ← ∞
α ← 1.0
do {
        S^α ← MSH-random(α)
        S ← S^α
        initialize(movalue, tabu_time)
        do {
                best_move ← find best(S)
                S ← execute(best_move)
                update(movalue, tabu_time)
                if (F(S) < F(S^α)) S^α ← S
        } while (moves without improving F(S^α) ≤ 50)
        if (F(S^α) < F(S*)) S* ← S^α
        α ← α - 0.25
} while (α ≥ 0)
```

Fig. 3. A *pseudo*-code for GTS

```
1   YS ← [π*_k(1), ..., π*_k(n_k)]
2   i ← n_k
3   π_k(i+1) ← n + k
4   CL(i) ← candidates(YS, i)
5   do {
6           while (CL(i) ≠ ∅ and i ≥ 1) {
7                   π_k(i) ← next branch(CL(i))
8                   CL(i) ← CL(i) - π_k(i)
9                   if (F_L < F(Π*_k)) {
10                          if (i = 1) {
11                                  Π*_k ← Π_k
12                                  F(Π*_k) ← F_L
13                          } else {
14                                  YS ← YS - π_k(i)
15                                  i ← i - 1
16                                  CL(i) ← candidates(YS, i)
17                          }
18                  }
19          }
20          YS ← YS ∪ π_k(i)
21          i ← i + 1
22          if (CL(i) ≠ ∅) YS ← YS ∪ π_k(i)
23  } while (i ≤ n_k)
```

Fig. 4. A *pseudo*-code for BBP

break ties between competing best solutions. If two trial schedules result in the same objective function value, the one with the smaller makespan is preferred. This rule is particularly important in multi-stage production systems for which the solution of the WE problem in one stage provides the deadlines (or due dates) for the preceding stage. Thus, the rule allows additional slack time to the early stages of the production system.

4. Branch-and-bound post-processor

The type of moves embedded in GTS allows for a quite extensive search through different job partitions, however the search is rather limited within a particular job assignment (note that swap moves of immediate neighbors are the only ones allowed within a machine). Therefore, it is likely for a solution to consist of a very good partition (possibly optimal) for which some sequences are not optimal. Since the number of jobs assigned to each machine is relatively small for the problems studied here ($n_k \approx 10$), a branch-and-bound post-processor (BBP) was created to find (or confirm) optimal job sequences on each machine. BBP is a procedure that may be applied to solutions at different levels, as follows:

Level	*Applied to*
1	S^* after all α values are explored.
2	S^α after local search is abandoned at the current α.
3	S after every move.

Level 2 includes 1, but 3 does not necessarily include 2. Using BBP at level 3 is computationally too expensive. The application at level 2 is beneficial only if after post-processing one of the inferior S^α solutions improves in such a way that it becomes better than the post-processed S^*. Since during preliminary experimentation this phenomenon was not observed, we opted for applying BBP at the first level.

Figure 4 presents a *pseudo*-code for BBP. The best solution found by the GTS procedure, S^*, consists of m sequences Π^*_k ($k = 1, \ldots, m$) and a set of completion times c^*, therefore it is required for BBP to be called m times. After each time, the best sequence for machine k, Π^*_k, is either confirmed to be optimal or replaced (in this case completion times for the jobs assigned to k are modified accordingly). As before, we define YS as the set of jobs yet to be scheduled which initially contains all jobs assigned to machine k (line 1). A variable i is used as a pointer to the current tree level being explored (levels are examined from the last position in the sequence to the first, see line 2). CL(i) is the list of unexplored branches at the ith level. This list is initialized every time the search moves to a lower level (lines 4 and 16). CL (i) is found as proposed by Chand and Schneeberger, (1988), where it was used for the reaching process of the DP algorithm.

$$T = \min \{c_{\pi_k(i+1)} - t_{\pi_k(i+1)}, \max_{j \in \mathrm{YS}} (d_j)\}$$

$$\Delta = \min_{j | d_j \geq T(t_j)}$$

$$\mathrm{CL}(i) = \{j | d_j > T - \Delta, j \in \mathrm{YS}\}$$

The value of Δ is used to allow idle time immediately before job $\pi(i+1)$. It is easy to verify that no optimal schedule contains an idle time greater than or equal to Δ.

Branches are selected for exploration (line 7) in increasing r-ratio values. Every time a branch is selected, it is deleted from the candidate list (line 8). The fathoming criterion consists of comparing the value of a lower bound F_L with the value of the incumbent solution (line 9). The lower bound is calculated as follows. Let $s = c_{\pi_k(i)} - t_{\pi_k(i)}$ be the starting-time at level i, then

$$F_L = \sum_{j=i}^{n_k} (d_{\pi_k(j)} - C_{\pi_k(j)})(p_{\pi_k(j)}) + \sum_{j|d_j>s} (d_j - s)\, p_j$$

If a branch is not fathomed (i.e. $F_L < F(\Pi_k^*)$), the search moves to a lower level. If the new level is the first position, the incumbent solution is updated (lines 11 and 12). Back-tracking is performed by the instructions in lines 20–22. The combined merit of GTS and BBP is assessed in the following section, where the result of computational experiments is presented.

5. Computational experiments

The GTS procedure is designed to seek an optimal solution to instances of the WE problem disregarding any assumptions about the r ratios or deadlines. It is known, for example, that MSH optimally solves the single machine case of the WE problem when for all jobs either the r ratios or the deadlines are equal (Chand and Schneeberger, 1988). However, for practical purposes, these assumptions are very unrealistic.

In the manufacturing environment that motivated our study, scheduling decisions are made one month in advance. This creates a 60-day planning horizon for which all deadlines fall within the last half of the period (i.e. $31 \leq d_j \leq 60$ for $j = 1, \ldots, n$). Deadlines also have the characteristic of being clustered around certain days of the week (specifically, from Wednesday to Friday). In an attempt to create instances that capture this particular demand behavior, we designed the following problem generator

$$d_j = \mathrm{U}(0,2) + 7\,(1 + \mathrm{U}(0,3)) + 30$$
$$t_j = \mathrm{U}(1,7)$$
$$p_j = \mathrm{U}(1, 2t_j)$$

Using the uniform distributions above, five sets of five problems were generated with the number of jobs ranging from 20 to 100. The number of machines was set equal to $n/10$ for all problems.

GTS requires (in addition to a seed for the random number generator) a value for the size of the short-term memory function (i.e. *tabu_size*). This value must be sufficiently large to avoid short-term cycling. In our case, $\lfloor\sqrt{n}\rfloor$ (where $\lfloor x\rfloor$ is the smallest integer greater than or equal to x) was found to be an acceptable lower bound on the value of tabu_size. In our context, tabu_size directly depends on the number of jobs, since moves are classified tabu as a result of the existence of tabu jobs. The maximum value for tabu_size is therefore $n-1$ (i.e. when only one job is not tabu at any particular time). The minimum value for tabu_size, when similar tabu restrictions are imposed, has been found to be in the neighborhood of 7 (Laguna, Barnes and Glover, 1990). Our choice of a lower bound for tabu_size is merely empirical, but it is in agreement with results reported in the literature.

An initial experiment was performed, in which tabu_size was set to its lower bound and the arbitrary seed value of 32164 was used, with the following goals in mind

(1) Measuring the merit of using the BB post-processor.

(2) Estimating the quality of the solutions obtained by GTS by comparing them with the solutions found by the MSH procedure.

(3) Estimating the average computational effort involved in a solution attempt for each problem size.

Tables 1 and 2 summarize the results of this experiment. Table 1 gives the percentage decrease in the total objective function (TOF) value over the MSH procedure, achieved by GTS with and without post-processor (the TOF value is simply the sum of the objective function value for each of the five problems in the set). Table 2 reports the CPU run times for these two procedures. The procedures were implemented in C and run on a 386/16 microcomputer. Table 1 shows that for problems with 20 and 40 jobs the best schedules found by GTS contained optimal job sequences on all machines. For larger problems ($n > 40$) the post-processor was able to improve upon the best GTS solu-

Table 1. Percentage decrease in the total objective function value

	Problem size				
Procedure	*20*	*40*	*60*	*80*	*100*
GTS	10.3	13.8	4.6	10.0	10.9
GTS/BBP	10.3	13.8	4.9	10.3	11.5

Table 2. Average CPU time in minutes

	Problem size				
Procedure	*20*	*40*	*60*	*80*	*100*
GTS	0.81	5.65	10.68	17.51	27.90
GTS/BBP	0.83	5.73	10.85	17.64	28.02

Table 3. Percentage deviation from best known solutions

	Solution attempt (a)								
Problem size (n)	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
20	0.06	0.00	0.06	0.06	0.06	0.06	0.06	0.23	0.46
40	0.54	0.89	0.74	0.31	0.70	0.46	0.39	0.23	0.50
60	3.04	2.75	2.63	2.20	2.78	1.19	2.12	2.16	1.25
80	1.73	1.49	2.09	1.76	1.51	1.62	1.48	2.09	1.92
100	2.61	2.67	2.98	2.61	2.25	2.42	2.26	2.29	2.80
Total	2.01	1.92	2.15	1.80	1.79	1.51	1.61	1.81	1.76

tions. The improvements are naturally due to the presence, within these solutions, of non-optimal job sequences. Table 2 shows that, for our test problems, the use of the post-processor did not result in a significant increase on CPU run time. Therefore, GTS/BBP was used for the remaining part of the computational experimentation. There also appears to be a non-exponential growth in the average CPU time required to acquire solutions as the number of jobs and machines grows larger.

A more extensive experiment was undertaken to determine a tabu_size value (or a set of values) that could be considered superior for the class of problems under study. The experiment consisted of 8 additional solution attempts to each problem instance using GTS/BBP. For each attempt, a, the short-term memory was set to

$$\text{tabu\_size} = \lfloor \sqrt{n} \rfloor + a \qquad \text{for } a = 1, \ldots, 8$$

Table 3 shows the percentage deviation between the TOF value obtained using a particular tabu_size and the best TOF value known for the given problem set (the best known solution for each problem is taken to be the minimum over all solution attempts). The last row in this table gives the total percentage deviation, calculated by adding the objective function values for all problem sizes. Note first that $(n, a) = (20, 1)$ is the only case where all solutions found corresponded to the best known solutions. The best memory size settings (i.e. a values) for problems with 40, 60, 80, and 100 jobs are 7, 5, 6, and 4, respectively. The most consistent memory size corresponds to an a value of 5, since overall it results in the smallest percentage deviation from the best known (i.e. 1.51%).

In order to confirm empirically the consistency of the solutions obtained by setting tabu_size to a value of $\lfloor \sqrt{n} \rfloor + 5$, a final experiment was performed. This experiment consisted of 6 solution attempts to each test problem of all sizes using the specified memory size and a random seed. The resulting solutions yielded a total deviation of 1.49% from the best known. This percentage deviation is very similar to the one reported in Table 1 for the same memory size, which strongly suggests the adequacy of the selected tabu_size value.

6. Conclusions and final remarks

The hybrid GRASP/tabu search approach with a branch-and-bound post-processor succeeds in finding solutions to the WE problem that are on the average at least 10% better than those found by the adapted Smith-heuristic. The solutions were found within a reasonable amount of time, considering the computer equipment utilized, and more importantly, taking into account that scheduling decisions (in the production environment we studied) are made once a month. In the case that the original schedule cannot be completed due to breakdowns, unexpected production orders, quality problems, or accidents, our solution method can be used to generate a revised solution to the problem. Partially completed jobs along with the new orders are then considered for scheduling purposes on a number of machines that might be adjusted to take into consideration reduction in capacity.

The use of BBP was only possible because of the 10-to-1 relationship between the number of jobs and machines in the set of test problems. If the ratio n/m grows, post-processing using BBP becomes rapidly infeasible (e.g. when the number of machines is set to 3 for the set of 40 job problems, the average CPU times for GTS and GTS/BBP become 5.9 and 12.4 min, respectively).

Within the scope of this research effort, we were unable to provide a stronger measure for the quality of our solutions (other than the one presented in Tables 1 and 3). Therefore, a natural direction for future research may consist of studying the properties of the MSH procedure as adapted to handle parallel identical machines, and the cases in which this approach performs well. In this way, a better understanding might be gained about the merit of applying GTS (with and without post-processing) to the multi-machine version of the weighted earliness problem.

References

Ahmadi, R. and Bagchi, U. (1986) Just-in-Time scheduling in single machine systems *Working Paper 85/86-4-21*, The University of Texas at Austin, USA.

Baker, K. R. and Scudder, G. D. (1990) Sequencing with earliness and tardiness penalties: a review. *Operations Research*, **38**, 22–36.

Chand, S. and Schneeberger, H. (1988) Single machine scheduling to minimize weighted earliness subject to no tardy jobs. *European Journal of Operational Research*, **34**, 221–30.

de Werra, D. and Hertz, A. (1989) Tabu search techniques: a tutorial and an application to neural networks. *OR Spectrum*, **11**, 131–41.

Feo, T. A. and Resende, M. G. C. (1989) A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, **8**, 67–71.

Glover, F. (1989) Tabu search—Part I. *ORSA Journal on Computing*, **1**, 190–206.

Glover, F. (1990) Tabu search—Part II. *ORSA Journal on Computing*, **1**, 4–32.

Glover, F. and Greenberg, H. J. (1989) New approaches for heuristic search: a bilateral linkage with artificial intelligence. *European Journal of Operations Research*, **39**, 119–30.

Laguna, M., Barnes, J. W. and Glover, F. (1990) Tabu search methods for a single machine scheduling problem, *Journal of Intelligent Manufacturing*, in press.

Laguna, M. and Glover, F. (1990) On target analysis and diversification in tabu search. *Technical Report*, University of Colorado at Boulder, USA.