



Theory and Methodology

Scheduling jobs on parallel machines with sequence-dependent setup times

Young Hoon Lee ^{a,*}, Michael Pinedo ^b

^a Samsung Electronics, Semiconductor Division, System Support Group, P.O. Box 37, Suwon 449-900, South Korea

^b Department of Industrial Engineering and Operations Research, Columbia University, New York, NY 10027, USA

Received 27 September 1994; accepted 23 November 1995

Abstract

Consider a number of jobs to be processed on a number of identical machines in parallel. A job has a processing time, a weight and a due date. If a job is followed by another job, a setup time independent of the machine is incurred. A three phase heuristic is presented for minimizing the sum of the weighted tardinesses. In the first phase, as a pre-processing procedure, factors or statistics which characterize an instance are computed. The second phase consists of constructing a sequence by a dispatching rule which is controlled through parameters determined by the factors. In the third phase, as a post-processing procedure, a simulated annealing method is applied starting from a seed solution which is the result of the second phase. In the dispatching rule of the second phase there are two parameters of which the values are dependent on the particular problem instance at hand. Through extensive experiments rules are developed for determining the values of the two parameters which make the priority rule work effectively. The performance of the simulated annealing procedure in the third phase is evaluated for various values of the factors. © 1997 Elsevier Science B.V.

Keywords: Scheduling; Heuristics; Parallel machines; Setup time

1. Introduction

Consider m identical machines in parallel with n jobs to be scheduled. The processing time, weight and due date of job j are denoted p_j , w_j and d_j , respectively. In order to process job k after job j , a setup time s_{jk} is required which depends on job j as well as on job k , but does not depend on the machine the two jobs are processed on. The completion time of job j is denoted by C_j , and the tardiness

of job j , which is defined as $\max(C_j - d_j, 0)$, by T_j . The objective is to find the sequence which minimizes the sum of weighted tardinesses of the n jobs, i.e., $\sum_j w_j T_j$. Because of the setup times there may exist an optimal schedule in which a machine remains idle, and an available job waits for processing on another machine on which it may incur a shorter setup time and be completed earlier. In this paper, however, we do not allow unforced idleness: If a machine is free, and there is a job waiting, then the machine is not allowed to remain idle. In various manufacturing environments, the tardiness is one of the important performance measures. Especially, the order-based products such as ASIC (Application Specific IC) and Mask-ROM in semiconductor in-

* Corresponding author.

Present address: Department of Industrial Systems Engineering, Yonsei University, 134, Shinchon-dong, Sudaemoon-ku, Seoul 120-749, South Korea.

dstry, are controlled in production line according to the due dates and the change-over or setup times. Each product receives certain level of priorities depending on the revenue or the importance of the customers. They have many identical machines on which products run. Semiconductor equipments usually need to be running continuously to keep the present specification.

It is well known that minimizing the sum of the weighted tardinesses on a single machine with all weights being equal is NP-hard [7] in the ordinary sense. Lawler [11] developed a pseudo-polynomial algorithm for the total tardiness problem on a single machine. When the jobs have different weights, the problem is strongly NP-hard [12]. Potts and van Wassenhove developed algorithms to solve the single machine problem without setups in a reasonable time when there is a limited number of jobs [17–20]. For a survey of the total weighted tardiness problem with no setups, see Abdul-Razaq, Potts and van Wassenhove [1]. Many heuristics which yield near-optimal solutions have been developed for the problem without setups. Some of the heuristics take the form of a dispatching rule which determines the value of a priority index for each job. Dispatching rules are designed so that the priority index for each job can be computed easily using the information available at any time. However, these dispatching rules are myopic and usually yield suboptimal solutions. The *Earliest Due Dates* (EDD) rule and the *Weighted Shortest Processing Time* (WSPT) rule are the simplest and most widely used rules. More sophisticated dispatching rules have been proposed for single machine as well as parallel machines, e.g., the COVERT rule by Carroll [5] and the ATC (Apparent Tardiness Cost) rule by Vepsalainen and Morton [26]. Lee et al. [13] proposed the ATCS (Apparent Tardiness Cost with Setups) rule for the single machine when there are sequence-dependent setup times between the jobs. The ATCS rule was developed through the enhancement of the Raman et al. rule [21].

A number of papers addressed the problem of minimizing the sum of the weighted tardinesses on parallel machines with zero setup times. Elmagraby and Park [8], Barnes and Brennan [4] and Vasilescu and Amar [25] suggested a branch and bound scheme to obtain optimal solutions. A dynamic programming

algorithm was developed by Dogramaci [6]. Arkin and Roundy [3] considered the weighted tardiness problem on parallel machines for the special case in which the weight is proportional to the processing time, and introduce the so-called *Earliest Gamma Date* heuristic.

One of the techniques applied to scheduling in order to obtain solutions close to optimal is *Simulated Annealing*. Simulated Annealing is basically a randomized local search method. Unlike a myopic dispatching rule, simulated annealing always has a positive probability of getting out of a local optimum in its search for the global optimum. Since Metropolis [16] first suggested this technique for combinatorial problems, many papers have subsequently reported successful applications of the technique in various areas [10,14,15,22–24]. Matsuo et al. [15] developed an efficient way to solve the weighted tardiness problem on a single machine through simulated annealing. Simulated annealing requires a seed solution to initiate the algorithm. Johnson et al. [10] and Matsuo et al. [15] have both shown that a good seed solution can decrease the computation time considerably. In other words, an effective heuristic has to be combined with simulated annealing in order to get a solution close to the optimal fast.

In this paper we suggest a three phase heuristic to minimize the sum of the weighted tardinesses when there are sequence-dependent setup times. The remainder of this paper is organized as follows: Section 2 describes the general framework of our three phase procedure. In Section 3, Section 4 and Section 5, we describe Phases 1, 2 and 3 respectively. Conclusions and a discussion about future research are presented in Section 6.

2. The framework of the heuristic

In this section we describe the general framework of the heuristic which consists of three phases. In the first phase, as a pre-processing procedure, we estimate the makespan and compute factors or statistics to characterize an instance. In the second phase, we construct a sequence using a modified version of the ATCS rule (modifications are necessary to make it suitable for the parallel machine environment). A statistical analysis is conducted to develop scaling parameters for the dispatching rule. In the third

phase, as a post-processing procedure, a simulated annealing technique is applied with, as a seed solution, the schedule obtained in the second phase. Various neighborhood structures and acceptance probabilities are tested for different values of factors. Fig. 1 shows the framework of three phase heuristic.

In the first phase we compute the factors or statistics which characterize the instance at hand. The factors which are associated with the due date, are the *due date tightness* factor τ and the *due date range* factor R .

The factor τ is defined as $\tau = 1 - (\bar{d}/C_{\max})$ where \bar{d} is the average of the due dates and C_{\max} is the makespan. It is easy to see that the schedule is tight if τ is close to 1. The due date range factor R , which is a crude measure of how spread out the due dates are, is defined as $R = (d_{\max} - d_{\min})/C_{\max}$ where d_{\max} (d_{\min}) is the maximum (minimum) value of the due dates. The *setup time severity* factor η which characterizes the effect of the setup times on the schedule was introduced by Lee et al. [13]. The factor η is defined as $\eta = \bar{s}/\bar{p}$ where \bar{s} is the average of the setup times and \bar{p} the average of the processing times. We consider a fourth factor which plays an important role in a parallel machine environment, the so-called *job-machine* factor μ . The factor μ which is defined as $\mu = n/m$ represents the average number of jobs to be processed on one machine.

The dispatching rule employed in the second phase is based on the ATCS rule. It is assumed that every job is available at time zero. A schedule for the parallel machines can be found as follows: Whenever one machine is freed, the following index is computed for each one of the unscheduled jobs:

$$I_j(t, l) = \frac{w_j}{p_j} \exp\left(-\frac{\max(d_j - p_j - t, 0)}{k_1 \bar{p}}\right) \times \exp\left(-\frac{s_{lj}}{k_2 \bar{s}}\right), \quad (1)$$

where $I_j(t, l)$ is the index for job j at time t given that job l is the last one completed on the machine just freed. At time t the job with the largest index is selected. If more than one machine is freed at one point in time, choose one arbitrarily. This dispatching rule is determined through the *look-ahead* or

PHASE 1	1. Find the values of factors, μ and η from the data of an instance. 2. Estimate the makespan 3. Find the values of factors, τ and R with the estimated makespan.
PHASE 2	1. Find the values of scaling parameters, k_1 and k_2 . 2. Apply the dispatching rule to generate the sequence.
PHASE 3	1. Decide the rule for the neighborhood structure. 2. Decide the form of the acceptance probability. 3. Apply the simulated annealing algorithm.

Fig. 1. Framework of the three phase heuristic.

scaling parameters, k_1 and k_2 . These parameters depend on the problem instance at hand, which can be characterized by the factors τ , R , η and μ . In Section 3, we address how to estimate the parameters, k_1 and k_2 as a function of these factors. Because of the setup times and the parallel machine environment, the makespan is not known until the schedule is known. We therefore construct an estimation procedure for the makespan, and with the estimated makespan we compute the values of the factors and proper values for k_1 and k_2 . Knowing k_1 and k_2 , we then apply the ATCS rule on the given instance and complete the second phase of the procedure.

The third phase is a post-processing procedure which is based on simulated annealing. A deterministic local search method consistently compares a new solution with the current best one, and takes the better one of the two. It may land in a local optimum which is far from a global optimum. Simulated annealing allows a shake out of a local minimum trap with a certain positive probability by allowing a move to an inferior solution. Let $G(x)$ be the value of objective function for the solution x . Let y be a so-called *neighborhood* solution of solution x . If solution y is better than solution x , i.e., $G(y) < G(x)$, we accept solution y as the current best. Otherwise, if $G(y) > G(x)$, we accept solution y with a certain probability, which is called the *acceptance probability*. For a minimization problem, Metropolis et al. [16] used the acceptance probability

$$P_{xy}(k) = \min\left(1, \exp\left(-\frac{G(y) - G(x)}{\alpha_k}\right)\right), \quad (2)$$

where k is the stage of the search and α_k are control parameters satisfying $\alpha_1 > \alpha_2 > \dots$. This α_k de-

creases to zero as k increases, which implies that the acceptance probability for a move to an inferior solution is smaller at a later stage in the process. From the definition of the acceptance probability it also follows that the worse a neighboring solution is, the smaller the acceptance probability is. The acceptance probability function remains the same within a stage. As the simulated annealing procedure converges to optimality with probability one after an infinite number of iterations, it requires in practice a large amount of computation time. Considerable research has been directed in methods for accelerating the convergence of simulated annealing. Greene and Supowit [9] made an attempt to find a good acceptance probability function by controlling the parameter α_k and the acceptance mechanism. Efforts for determining appropriate neighborhood structures have been made by Matsuo et al. [15]. They also showed that good initial solutions reduce the computation time considerably and also lead to good final solutions.

A simulated annealing heuristic requires the number of stages, and the number of iterations at each stage to be specified, as well as the acceptance probability function at each stage and the termination rule. A search can be made at each stage in a random way or in an organized way. The structure of the neighborhood is a design issue. In practice several stopping criteria can be used. One way is to specify the number of iterations and stages. Another is to let the procedure run until for a given number of iterations no improvement has been obtained.

The general scheme of a simulated annealing algorithm is as follows:

Step 0: Initialization

I = total number of stages

J = number of iterations for searches at each stage

$P_{xy}(k)$ = acceptance probability at stage k
 $stage = 0$
 $iteration = 0$

Step 1: x = an initial solution obtained by the ATCS rule

Step 2: $stage = stage + 1$. If $stage > I$, then terminate.

Step 3: $iteration = iteration + 1$. If $iteration > J$, then go to Step 2.

Step 4: Find a neighbor of x , say y .

Step 5: If $G(y) < G(x)$, then go to Step 6, else Step 7.

Step 6: Set $x = y$, keep x as current optimum. Go to Step 4.

Step 7: $random =$ a random number generated from the uniform distribution on $[0,1]$. If $random \leq P_{xy}(k)$, then $x = y$ and go to Step 3, else go to Step 3.

We experimented with various types of acceptance probabilities and various stopping criteria in terms of number of iterations and stages. The results are reported in Section 4.

3. The first phase: Pre-processing procedure

In Phase 1 we find values for the factors or statistics which are needed to determine the scaling parameters for the dispatching rule. In order to obtain values for the factors we need an estimate of the makespan for the given instance. Assuming that the schedule is generated by ATCS we use the following estimator for the makespan:

$$\hat{C}_{\max} = (\beta \bar{s} + \bar{p})\mu, \quad (3)$$

where β is a coefficient which takes into account the effect of the setup times on the makespan. The average time to process a job including its setup time is assumed to be $(\beta \bar{s} + \bar{p})$ with $\beta \leq 1$, see Lee et al. [13] for details. The estimator \hat{C}_{\max} is also based on the assumption that under a good schedule all machines are likely to become idle at approximately the same time. We performed extensive experiments to find an appropriate value for the coefficient β . The way the job data are generated is described in Section 4. We start with $\beta = 0.5$ and determine proper values for parameters k_1 and k_2 . With these k_1 and k_2 we conduct an experiment in order to obtain a better value for β and so on. It is easy to see that the makespan depends on η and μ . From the experiments we concluded that β is sensitive to the factors η and μ and not that sensitive to the values of k_1 and k_2 . We suggest the following function to estimate β for μ sufficiently large (e.g., $\mu \geq 5$):

$$\beta = 0.4 + \frac{10}{\mu^2} - \frac{\eta}{7}. \quad (4)$$

Table 1
Comparison of the makespans for β

μ	$\hat{\beta}$	\hat{C}_{\max}	C_{\max}	Error (%)
5	0.73	682.1	694.3	1.76
10	0.43	1214.3	1167.8	3.98
15	0.37	1779.8	1747.7	1.83
20	0.35	2353.6	2363.3	0.41
30	0.34	3509.5	3511.7	0.06
60	0.33	6994.1	6558.8	6.64

Given $\tau = 0.7$, $R = 0.5$, $\eta = 0.5$, Error = $100|C_{\max} - \hat{C}_{\max}|/C_{\max}$ and the data are generated in the way described in Section 4.

In Table 1, the actual makespan is compared with the estimated makespan using the recommended values for β under the above rule for each value of μ . These errors observed are typical for different values of the various factors. We studied various other, slightly more sophisticated, estimators for the makespan, but none of these resulted in better estimates.

It is clear that the makespan of the schedule also depends on the variability in the setup times. Define the coefficient of the variation c_v as $\text{Var}(s)/\bar{s}^2$ where $\text{Var}(s)$ is the variance of setup times. Throughout the study we assume that c_v is $1/3$, which is the value of c_v for the uniform distribution on $[0, u]$ with positive value u (See Section 4.) Fig. 2 shows how the best estimate for β changes as a function of c_v and μ in parallel machines.

4. The second phase: Dispatching rule

The effectiveness of the ATCS rule in Phase 2 depends on the scaling parameters k_1 and k_2 . In this

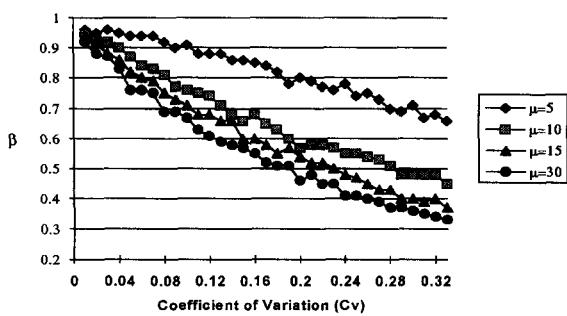


Fig. 2. Changes of β with respect to c_v and μ .

section we describe the experiments performed to determine the functions which map the factors of an instance into values for the parameters k_1 and k_2 . The number of jobs is set to 60 throughout the study. We designed the experiment as follows:

We generate 560 different combinations of factors, τ , R , η and μ , namely:

$$\tau = (0.3, 0.4, 0.5, \dots, 0.9),$$

$$R = (0.25, 0.5, 0.75, 1.0),$$

$$\eta = (0.25, 0.5, 0.75, 1.0),$$

$$\mu = (5, 10, 15, 20, 30).$$

For each combination, ten different instances are generated randomly. The processing time p_j of job j is generated from the uniform distribution over the interval $[50, 150]$. The mean, \bar{p} , is 100. The due date d_j of job j is generated using the two factors τ and R . The due date is uniformly distributed over the interval $[(1 - R)\bar{d}, \bar{d}]$ with probability τ and uniformly distributed over the interval $[\bar{d}, \bar{d} + (C_{\max} - \bar{d})R]$ with probability $(1 - \tau)$. It is easy to see that the average of the due dates generated is \bar{d} . The weight w_j of job j is generated from the uniform distribution on $[0, 10]$. The generation of the setup time is controlled by the factor η . Setup times are uniformly distributed over the interval $[0, 0.2\bar{s}]$, or $[0, 2\eta\bar{p}]$.

For 5600 instances, 10 different instances for 560 different types, the ATCS rule is applied for different values of k_1 and k_2 :

$$k_1 = (0.2, 0.4, 0.6, \dots, 6.4),$$

$$k_2 = (0.1, 0.2, 0.3, \dots, 1.6).$$

For each value of k_1 and k_2 , the value of objective function, i.e., the sum of the weighted tardinesses, is computed. Let Z be the minimum value of the objective functions computed. All k_1 and k_2 values which lead to values of the objective function which are less than or equal to $(1 + \alpha Z)$ are identified and the average is taken for both k_1 and k_2 . The value of α is a function of τ , namely $\alpha = 0.095 - \tau/10$. We introduce the α so that a certain proportion of the values near minimum objective function are to be considered. If τ is high, i.e., the schedule is tight, then there is less variability in the value of the objective function, which suggest a lower α . For

given values of τ , R , η and μ , the search for the best k_1 and k_2 is performed with ten instances and the average is taken as the best estimate of parameters. From the results of the experiment we conclude the following:

1. The k_1 value is increasing in τ , η and μ , and decreasing in R . The k_1 value is very sensitive to R and μ , moderately sensitive to η .
2. The k_2 value is increasing in τ , and decreasing in η and μ . The k_2 value is very sensitive to τ and η , moderately sensitive to μ and invariant to R .

Fig. 3 and Fig. 4 show the general structure of k_1 and k_2 with respect to the factors which affect most for each one. The following rules could be used for the selection of proper values for k_1 and k_2 :

$$k_1 = 1.2\ln(\mu) - R, \quad k_2 = \frac{\tau}{A_2\sqrt{\eta}}, \quad (5)$$

where $A_2 = 1.8$ if $\tau < 0.8$, $A_2 = 2.0$ if $\tau \geq 0.8$. The following modification has to be made:

1. Subtract 0.5 from k_1 if $\tau < 0.5$,
2. Subtract 0.5 from k_1 if $\eta < 0.5$ and $\mu > 5$.

In order to measure the performance of parameters obtained in Phase 2, we test how good the schedule generated by the ATCS rule is. The sum of the weighted tardiness under the ATCS schedule is compared with the sum of the weighted tardiness under the optimal schedule. The optimal schedule can be found only for problems of a small size because of the computation time involved. We compared instances with 10 jobs and 2 machines. It has to be noted that the variety of the instances generated is important in order to appreciate the effectiveness of the heuristic. In particular, the tightness of an

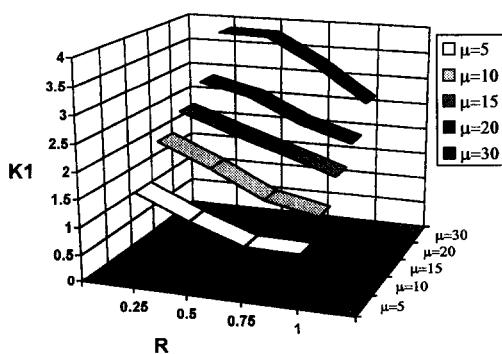


Fig. 3. Plot of k_1 as a function of μ and R .

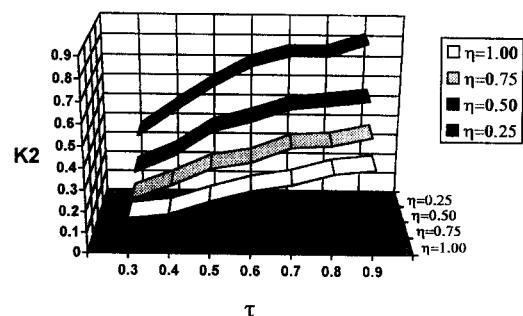


Fig. 4. Plot of k_2 as a function of τ and η .

instance is a critical factor in deciding the efficiency of a heuristic for the total weighted tardiness problem. For various values of τ , R and η , as suggested earlier, we generate the appropriate instance and find the optimal schedule by enumeration.

We can measure the performance by computing the difference between the ATCS schedule's total weighted tardiness, and the optimal schedule's. Let the so-called *Relative Error* (RE) be defined as follows:

$$RE = \frac{WT(ATCS) - WT(OPT)}{WT(OPT)}, \quad (6)$$

where $WT(ATCS)$ ($WT(OPT)$) denotes the sum of the weighted tardinesses under the ATCS (optimal) schedule. This measure can be found only for instances in which WT of the optimal schedule is greater than zero. Even for instances with a small value of WT in the optimal schedule, a small deviation may result in a big ratio even if the schedule is not far from the optimal.

We developed another measure called *Normalized Relative Error* (NRE), which reflects the data of an instance such as due dates and processing times, to be defined as follows:

$$NRE = \frac{WT(ATCS) - WT(OPT)}{n\bar{w}\tau^2\hat{C}_{max}/2}, \quad (7)$$

where \bar{w} is the average weight of tardy jobs, and \hat{C}_{max} is the estimated makespan. The normalization factor $n\bar{w}\tau^2\hat{C}_{max}/2$ can be interpreted as a measure for the expected WT of the instance since the average due date is $(1 - \tau)\hat{C}_{max}$, the average number of tardy jobs is approximately $n\tau$, and the average

Table 2
Performance of ATCS schedules: NRE(ARE)

η	τ	$R = 0.25$		$R = 0.5$		$R = 0.75$		$R = 1.0$	
		NRE	ARE	NRE	ARE	NRE	ARE	NRE	ARE
0.25	0.3	0.2385	–	0.5252	–	0.6983	–	0.6342	–
	0.5	0.0861	–	0.2015	–	0.0537	–	0.0987	–
	0.7	0.0606	0.0648	0.0295	0.0224	0.1060	0.0721	0.0850	0.0946
	0.9	0.0190	0.0215	0.0123	0.0084	0.0270	0.0230	0.0404	0.0528
0.50	0.3	0.2193	–	0.6328	–	0.9918	–	0.1482	–
	0.5	0.2337	–	0.4018	–	0.4394	–	0.0435	–
	0.7	0.1152	0.0776	0.0386	0.0421	0.0875	0.0923	0.1121	0.0901
	0.9	0.0899	0.0743	0.0473	0.0297	0.0585	0.0500	0.0382	0.0240
0.75	0.3	0.4402	–	0.6058	–	0.2842	–	0.2390	–
	0.5	0.3812	–	0.1590	–	0.1380	–	0.1735	–
	0.7	0.2725	0.5304	0.2027	0.2114	0.2466	0.3872	0.2040	0.2195
	0.9	0.1040	0.0859	0.0931	0.0696	0.1412	0.0956	0.0505	0.0414
1.00	0.3	0.4125	–	0.9614	–	0.2842	–	0.3088	–
	0.5	0.3224	–	0.0808	–	0.2358	–	0.0514	–
	0.7	0.3233	0.2862	0.2329	0.2146	0.0963	0.1658	0.1184	0.1526
	0.9	0.0000	0.0000	0.0803	0.0373	0.0407	0.0305	0.0857	0.0479

delay of tardy jobs is $\tau \hat{C}_{\max}/2$. In our experiment, \bar{w} is set to 3, since it is expected that the jobs which are allowed to be tardy in a good schedule are likely to have a low weight. We can apply this measure for any instance because the normalization factor is always positive. The *Average Relative Error* (ARE) which is the average RE for ten instances, is found only for $\tau \geq 0.6$ in which every instance generated in our experiment has positive WT.

Table 3
Average NRE(ARE) for τ , R and η

τ	NRE	ARE
0.3	0.4765	–
0.5	0.1938	–
0.7	0.1457	0.1858
0.9	0.0580	0.0432
R	NRE	ARE ^a
0.25	0.2074	0.1426
0.50	0.2691	0.1046
0.75	0.2456	0.1146
1.00	0.1520	0.0904
η	NRE	ARE ^a
0.25	0.1823	0.0450
0.50	0.2311	0.0600
0.75	0.2335	0.2051
1.00	0.2272	0.1169

^a Average values for instances with $\tau \geq 0.6$.

Table 2 shows the performance of the ATCS rule for various values τ , R and η . For a high value of τ , the ATCS rule works well. If τ is greater than or equal to 0.7, and η is less than or equal to 0.5 (i.e., setup times are relatively small in comparison with processing times), the objective functions of the schedules obtained by the ATCS rule are at most 9% away from the optimal schedule's. It is to be expected that obtaining a good solution for large η is harder as the effects of the setup times become more dominating. Notice that ARE is almost equal to NRE for $\tau \geq 0.6$ when the objective function in every instance has a positive value. Table 3 shows the average performance for each value of τ , R and η . It is clear that the performance of the heuristic is not that good for small values of τ , when the schedule is loose and the value of objective function is small.

5. The third phase: post-processing procedure

In Phase 2, we obtain a specific sequence for an instance. In Phase 3, we attempt to improve the sequence obtained in Phase 2 within a limited amount of computation time. In this section we describe a simulated annealing algorithm which uses the sequence obtained in Phase 2 as a seed solution. We

present a neighborhood structure for the parallel machine environment and two rules for the neighborhood search. We also test performances for various types of acceptance probabilities. As a termination rule we fix the number of iterations and stages.

In what follows the sequence refers to the order in which the jobs start with their processing. In a parallel machine environment, a neighborhood structure obtained by simple adjacent pairwise interchanges in the sequence may not lead to a better sequence. Let π_i be the sequence $(1, 2, \dots, i, i+1, \dots, n)$, and π'_i be the sequence obtained by interchanging jobs i and $i+1$, i.e., $(1, 2, \dots, i+1, i, \dots, n)$. Then the job sequence on each machine after job i under π'_i may be very different from the job sequence after job i under π_i . This may lead to a significant increase in the total setup times. Therefore the neighborhood has to be constructed in such a way that machine sequences with good setup times are maintained.

Suppose that a neighbor is constructed by interchanging two jobs, i and j which are adjacent in the sequence. (In fact, throughout this study a neighboring solution is obtained only through interchanges of two adjacent jobs.) If jobs i and j are processed on the same machine in the original sequence, the neighbor can be found easily: the two jobs are interchanged on the same machine and all other jobs are not changed. If jobs i and j are processed on different machines, a neighbor can be found as follows: all jobs following job i on its machine are interchanged with all jobs following job j on its machine. In this case, the sequence of the neighbor, which are ordered with respect to the starting times of the jobs, is usually very different from the original's. This interchange may give rise to a schedule where at some time during the schedule there is an unscheduled job and one machine is free, which is prohibited by the assumption (see Section 1). Therefore we have to adjust the sequence by moving jobs which are sequenced at the end on one machine to the other machine. But this adjustment may increase setup times, and hence completion times of jobs involved. If only adjacent jobs (not adjacent jobs on the same machine, but adjacent in the sequence) are to be interchanged, the start times are similar and there are unlikely to be many adjustments of this type.

The speed of a simulated annealing procedure depends on how fast the algorithm finds a good neighbor, which may lead to an optimal or near-optimal solution. When moving in a search from one solution to another, we have many alternatives for pairwise interchanges, i.e., $(n-1)$ possibilities in a n -job instance. If one pair of adjacent jobs have a large setup time in-between, then interchanging these two jobs can lead to a larger improvement than other interchanges. We take this point into account in our search for a good neighbor; this accelerates the algorithm significantly. The two jobs in a sequence to be interchanged are selected sequentially. At any stage and iteration, the job with the largest setup time is selected among jobs not yet considered.

Another important element of simulated annealing is the acceptance probability. As mentioned before, a positive value of the acceptance probability $P_{xy}(k)$ allows us to consider an inferior neighbor as a candidate. Such a candidate may eventually lead to an optimal or near-optimal solution. The $P_{xy}(k)$ is decreasing in k so that it lowers the probability of getting out of a local optimum at a later stage. Matsuo et al. [15] and Vakharia et al. [22] considered an acceptance probability function which is independent of the solutions x and y . It does not depend on the improvement achieved with the move. They have shown that the performances obtained using acceptance probabilities independent of the change are nearly as good as the performances obtained using acceptance probabilities which do depend on the change. Here we also consider similar types of functions which do not change within a stage. Various function types can be considered in terms of two characteristics: the initial acceptance probability and its functional form. In our experiment, the functional form of the acceptance probability selected is the linearly decreasing function with initial probability 0.6. The number of stages to be carried out is fixed at 20. Therefore, the acceptance probability decreases at every stage by 0.03. If a sequence is given a seed solution, the job with which the neighbor can be found through an interchange with the preceding job is selected by the amount of setup time incurred.

No practical algorithm has been developed yet for solving the 60-job problem optimally. The relative improvement of simulated annealing over the ATCS heuristic is therefore measured rather than the abso-

lute or relative error with respect to the optimal schedule. Let *Average Relative Improvement* (ARI) be defined as

$$\text{ARI} = \frac{\text{WT}(\text{SA})}{\text{WT}(\text{ATCS})}, \quad (8)$$

where $\text{WT}(\text{SA})$ denotes the sum of the weighted tardinesses of the schedule obtained after conducting a fixed number of searches at every stage. Table 4 shows performances of simulated annealing for different values of the number of searches in each stage, for $J = (5, 10, 20, 60, 180, 360)$. Experiments are performed on a SUN-SPARC workstation using the C programming language, and the computation times (in seconds) are measured for various number of stages.

As expected, a bigger improvement can be achieved when the number of searches increases. The computation time increases almost linearly with the number of searches. It can be observed that, for $J \geq 60$ no significant improvement can be obtained for any combination of factors. Therefore it might be said that the system approaches the state in which improvement becomes insignificant with respect to

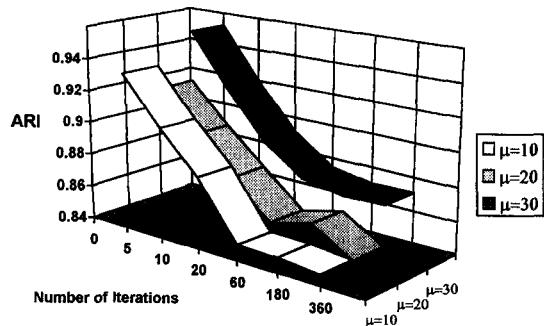


Fig. 5. Performance of Simulated Annealing.

the increase of the number of searches, in around 60 or 180 searches, which is almost equal to one or three times the number of jobs. In fact, in most applications of simulated annealing on scheduling problems reported in the literature, it is noted that the number of searches needed to reach such a state is at least 8 to 16 times the number of jobs [15]. In our study searches are however conducted systematically rather than randomly, i.e., the job with the largest setup time is selected for a move. This appears to accelerate the procedure (Fig. 5).

Table 4
Performance of Simulated Annealing (ARI)

J	5	10	20	60	180	360
CPU(sec)	1.2921	2.4029	4.6260	13.2397	38.7743	74.5820
μ	5	0.9179	0.8989	0.9007	0.8506	0.8496
	10	0.9348	0.9060	0.8826	0.8495	0.8489
	15	0.9384	0.9035	0.8823	0.8624	0.8562
	20	0.9205	0.8959	0.8757	0.8551	0.8612
	30	0.9523	0.9200	0.8912	0.8744	0.8705
	τ	0.1	0.6728	0.5767	0.5244	0.4336
	0.3	0.9247	0.8935	0.8616	0.8087	0.8081
	0.5	0.9913	0.9866	0.9832	0.9711	0.9719
	0.7	0.9988	0.9984	0.9979	0.9957	0.9946
	0.9	0.9995	0.9991	0.9987	0.9983	0.9983
	R	0.25	0.9503	0.9347	0.9167	0.8832
	0.50	0.9189	0.8943	0.8740	0.8384	0.8382
η	0.75	0.9111	0.8711	0.8534	0.8300	0.8245
	1.00	0.9508	0.9193	0.9027	0.8821	0.8841
	0.25	0.9296	0.9011	0.8860	0.8588	0.8568
	0.50	0.9260	0.8970	0.8802	0.8497	0.8464
	0.75	0.9369	0.9013	0.8851	0.8644	0.8603
	1.00	0.9386	0.9200	0.8955	0.8608	0.8655
Average		0.9328	0.9049	0.8874	0.8584	0.8573
						0.8545

6. Conclusions

This paper presents a dispatching rule and simulated annealing procedure for the total weighted tardiness problem on parallel machines when there are sequence-dependent setups. The dispatching rule appears to work very well, especially when the schedule is tight, with parameters suggested that depend on the instance data. The solution obtained with the dispatching rule is used as seed solution for the simulated annealing procedure. A conventional simulated annealing procedure selects a neighbor arbitrarily with a high acceptance probability. Therefore it requires often a large amount of computation time. We used the information which can be observed in any sequence, to find the best candidate for next neighbor. A job with large setup time in the sequence is selected to generate the neighborhood, which may bring about significant improvement in the simulated annealing procedure. This approach decreases the number of searches significantly needed to obtain a desired improvement.

In our ATCS rule the parameters are determined at time zero by the data of instance at hand, and do not change through the sequencing process. Since the parameters are functions of the data available, they can be changed as scheduling is going on. At time t , after some jobs are scheduled, we can recompute the parameters using the data of unscheduled jobs and can apply the ATCS rule with the updated parameters. Dynamic setting of parameters can be done periodically or whenever a job is sequenced.

Jobs with release dates may be considered. If jobs are released dynamically and continuously, the ATCS dispatching rule is easy to apply. Whenever there is any change in the set of available jobs, either through the release of new jobs or through the completion of old jobs, the parameters of the ATCS rule have to be updated.

The ATCS rule and simulated annealing can be adapted to be suitable for the total weighted tardiness problem without setup times. We then have one parameter k which can be evaluated easily. In the simulated annealing procedure, the large penalty criterion can be used. Interchange of jobs can be carried out more easily than in the case with setup times.

In our study we considered the case in which setup times are completely random. Setup times may

have a special structure, e.g., they may satisfy the triangular inequality. With regard to the setup times, we considered only the severity factor η , which measures the mean of setup times. The variability of setup times may affect the results as well. The algorithm can be applied to more complex problems after some modification which is necessary due to differences in the problem structure. Variations of the algorithm described in this paper have been used in a number of factories [2]. One of those is the BPSS scheduling system, which is implemented in a liquid packaging industry and results in schedules which are quite satisfactory.

References

- [1] Abdul-Razaq, T.S., Potts, C.N. and van Wassenhove, L.N., 1990. A Survey of Algorithms for the Single Machine Total Weighted Tardiness Scheduling Problem. *Discrete Applied Mathematics*, 26, 235–253.
- [2] Adler, L., Fraiman, N.M., Kobacker, E., Pinedo, M., Plotnickoff, J.C. and Wu, T.P., 1993. BPSS: A Scheduling System for the Packaging Industry. *Operations Research* 41, 641–648.
- [3] Arkin, E.M. and Roundy, R.O., 1991. Weighted-Tardiness Scheduling on Parallel Machines with Proportional Weights. *Operations Research*, 39, 64–81.
- [4] Barnes, J.W. and Brennan, J.W., 1977. A Improved Algorithm for Scheduling Jobs on Identical Machines. *IIE Transactions*, 9, 25–31.
- [5] Carroll, D.C., 1965. Heuristic Sequencing of Jobs with Single and Multiple Components. *Ph.D. Thesis*. Sloan School of Management, MIT.
- [6] Dogramaci, A., 1984. Production Scheduling of Independent Jobs on Parallel Identical Machines. *International Journal of Production Research*, 16, 535–548.
- [7] Du, J. and Leung, J.Y., 1990. Minimizing Total Tardiness on One Machine is NP-hard. *Mathematics of Operations Research*, 15, 483–494.
- [8] Elmaghraby, S.E. and Park, S., 1974. Scheduling Jobs on A Number of Identical Machines. *IIE Transactions*, 6, 1–13.
- [9] Greene, J.W. and Supowit, K.J., 1986. Simulated Annealing without Rejected Moves. *IEEE Trans. on Computer-aided Design, CAD-5*, 221–228.
- [10] Johnson, D.S., Aragon, C.R., McGeoch, L.A. and Schevon, C., 1989. Optimization by Simulated Annealing: An Experimental Evaluation; Part 1, Graph Partitioning. *Operations Research*, 37, 865–892.
- [11] Lawler, E.L., 1977. A Pseudo-Polynomial Algorithm for Sequencing Jobs to Minimize Total Tardiness. *Annals of Discrete Mathematics*, 1, 331–342.
- [12] Lawler, E.L., Lenstra, J.K. and Rinnooy Kan, A.H.G., 1982. Recent Developments in Deterministic Sequencing and

- Scheduling. In: M.A.H. Dempster, J.K. Lenstra and A.H.G. Rinnooy Kan, A.H.G., (eds.), *A Survey, Deterministic and Stochastic Scheduling*. Reidel, Dordrecht, 35–73.
- [13] Lee, Y.H., Bhaskaran, K. and Pinedo, M., to appear. A Heuristic to Minimize the Total Weighted Tardiness with Sequence Dependent Setups. *IIE Transactions*.
- [14] Matsuo, H., Suh, C.J. and Sullivan, R.S., 1988. A Controlled Search Simulated Annealing Method for the General Jobshop Scheduling Problem. *Working Paper 03-04-88*, Department of Management, The University of Texas at Austin.
- [15] Matsuo, H., Suh, C.J. and Sullivan, R.S., 1989. A Controlled Search Simulated Annealing Method for the Single Machine Weighted Tardiness Problem. *Annals of Operations Research*, 21, 85–108.
- [16] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A. and Teller, E., 1953. Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21, 1087–1092.
- [17] Potts, C.N. and van Wassenhove, L.N., 1982. A Decomposition Algorithm for the Single Machine Total Tardiness Problem. *Operations Research Letters*, 1, 177–182.
- [18] Potts, C.N. and van Wassenhove, L.N., 1985. A Branch and Bound Algorithm for the Total Weighted Tardiness Problem. *Operations Research*, 33, 363–377.
- [19] Potts, C.N. and van Wassenhove, L.N., 1987. Dynamic Programming and Decomposition Approaches for the Single Machine Total Tardiness Problem. *The European Journal of Operational Research*, 32, 405–414.
- [20] Potts, C.N. and van Wassenhove, L.N., 1991. Single Machine Tardiness Sequencing Heuristics. *IIE Transactions*, 23, 346–354.
- [21] Raman, N., Rachamadugu, R.V. and Talbot, F.B., 1989. Real-Time Scheduling of an Automated Manufacturing Center. *The European Journal of Operational Research*, 40, 222–242.
- [22] Vakharia, A.J. and Chang, Y., 1990. A Simulated Annealing Approach to Scheduling a Manufacturing Cell. *Naval Research Logistics*, 30, 559–577.
- [23] van Laarhoven, P.J.M. and Aarts, E.H.L., 1987. Simulated Annealing: Theory and Applications. D. Reidel, Dordrecht.
- [24] van Laarhoven, P.J.M., Aarts, E.H.L. and Lenstra, J.K., 1992. Job Shop Scheduling by Simulated Annealing. *Operations Research*, 40, 113–125.
- [25] Vasilescu, E.N. and Amar, A.D., 1983. An Empirical Evaluation of the Entrapment Procedure for Scheduling Jobs on Identical Machines. *IIE Transactions*, 15, 261–263.
- [26] Vepsäläinen, A. and Morton, T., 1987. Priority Rules for Job Shops with Weighted Tardiness Costs. *Management Science*, 33, 1035–1047.