Improved heuristics for the single machine scheduling problem with linear early and quadratic tardy penalties

Jorge M. S. Valente*
LIAAD – INESC Porto LA, Faculdade de Economia, Universidade do Porto
Postal Address: Faculdade de Economia, Universidade do Porto, Rua Dr. Roberto Frias, 4200-464 Porto, Portugal
E-mail Address: jvalente@fep.up.pt
*Corresponding author

Jeffrey E. Schaller
Department of Business Administration, Eastern Connecticut State University
Postal Address: Department of Business Administration, Eastern Connecticut State University, 83 Windham St., Willimantic, CT 06226-2295, USA
E-mail Address: schallerj@easternct.edu

Abstract

This paper considers the single machine scheduling problem with linear earliness and quadratic tardiness costs. The research on the version with inserted idle time focused on an exact approach, while several heuristics had already been proposed for the version with no idle time. These heuristics were then the basis for the development of new heuristic procedures for the version with idle time. Some improvement procedures were also considered.

The new heuristics outperformed the existing procedures. A genetic algorithm provides the best results in terms of solution quality, but is computationally intensive. One of the backward scheduling dispatching rules provides results of similar quality, and can quickly solve even large instances.

The new heuristics were also applied, with the appropriate modifications, to the version with no idle time. Again, the new procedures provided better results than the existing heuristics. Therefore, the procedures developed in this paper are the new heuristics of choice for both versions of the considered problem.

Keywords: production scheduling; heuristics; single machine; linear earliness; quadratic tardiness; dispatching rules; genetic algorithms

Biographical Notes

Jorge M. S. Valente is an Associate Professor of Operations Research at the Faculty of Economics, University of Porto, Portugal. He holds a Ph.D. in Management Science and a M.S. in Economics from the University of Porto. His current research interests include production scheduling, combinatorial optimization and heuristic techniques.

Jeffrey E. Schaller is a Professor of Operations Management at Eastern Connecticut State University. He holds a Ph.D. in Business Administration from the University of Florida, a Masters of Business Administration from the University of Michigan and a B.A. from Gettysburg College. His current research interests include production scheduling, cellular manufacturing and lean production.

1. Introduction

This paper considers the single machine scheduling problem with linear earliness and quadratic tardiness costs. Formally, the problem can be stated as follows. A set of n independent jobs $\{1, 2, \ldots, n\}$ has to be scheduled on a single machine that can handle at most one job at a time. The machine is assumed to be continuously available from time zero onwards, and preemptions are not allowed. Job j, $j = 1, 2, \ldots, n$, requires a processing time p_j and should ideally be completed on its due date d_j. For a given schedule, the earliness and tardiness of job j are respectively defined as $E_j = \max\{0, d_j - C_j\}$ and $T_j = \max\{0, C_j - d_j\}$, where C_j is the completion time of job j. The objective is then to find a schedule that minimizes the sum of the linear earliness and quadratic tardiness costs $\sum_{j=1}^{n}\left(E_j + T_j^2\right)$.

Single machine scheduling environments actually occur in several practical operations; for a specific example in the chemical industry, see Wagner, Davis and Kher (2002). Also, the performance of many production systems is frequently determined by the quality of the schedules for a single bottleneck machine. Moreover, results and insights obtained for single machine problems can often be applied to more complex scheduling environments, such as flow shops or job shops.

Scheduling models with both earliness and tardiness costs have received considerable and increasing attention from the scheduling community, due to their practical importance. Indeed, early/tardy scheduling problems are compatible with the concepts of just-in-time production and supply chain management. These production strategies, which have been adopted by many organisations, view both early and tardy deliveries as undesirable.

The objective function considered in this paper includes linear earliness and quadratic tardiness costs. Indeed, early deliveries result in unnecessary inventory, and the costs incurred with this inventory tend to be proportional to the time held in stock. Late deliveries, on the other hand, can result in lost sales and loss of customer

goodwill. A quadratic penalty is then used for the tardy jobs, since a customer's dissatisfaction tends to increase quadratically with the tardiness, as proposed in the loss function of Taguchi (1986).

The considered problem is of practical relevance and interest when idle time may be inserted, as well as when the insertion of idle time is not allowed. Both of these versions of the problem have been previously studied. Schaller (2004) analysed the problem with inserted idle time. He presented a timetabling algorithm and a lower bounding procedure, as well as a branch-and-bound algorithm and simple heuristics. For the problem with no idle time, both an exact procedure and several heuristic techniques have been considered. Valente (2008) developed a lower bounding procedure and a branch-and-bound technique. Among the heuristics, dispatching rules (Valente, 2007b), beam search procedures (Valente, 2007a) and genetic algorithms (Valente and Gonçalves, 2008) have all been proposed.

A large number of papers have been published on scheduling models with earliness and tardiness costs. Baker and Scudder (1990) provide an excellent survey of the initial work on early/tardy scheduling. A recent survey of multicriteria scheduling which includes problems with earliness and tardiness penalties is given in Hoogeveen (2005), while Kanet and Sridharan (2000) review scheduling models with inserted idle time.

Previous research proposed a large number of heuristics for the problem with no idle time. Among these heuristics, the EQTP_EXP dispatching rule (henceforth denoted simply as EQTP) developed in Valente (2007b) is the heuristic of choice for large instances, while the MA_IN genetic algorithm of Valente and Gonçalves (2008) provides the best results for small and medium size problems. This paper investigates heuristic procedures for the version with inserted idle time, taking as its starting point the existing heuristics for the problem where idle time is not allowed. In this context, the performance of the EQTP dispatching rule, suitably adapted in order to take into account the insertion of idle time, was first analysed. Since the results were not as good as desired, several additional dispatching heuristics were then developed, as well as a modification of the genetic algorithm MA_IN. Some improvement procedures were also considered, in order to improve the schedule obtained by the heuristics.

These new heuristics performed quite well for the problem with inserted idle time. Therefore, this raised the question of whether they would also perform well for the version with no idle time. Consequently, the new heuristics were also then

applied, after the appropriate modifications, to the problem without idle time. These new procedures outperformed the existing heuristics, so the procedures developed in this paper are the new heuristics of choice for both versions of the considered problem.

The remainder of this paper is organized as follows. In section 2, the heuristic procedures are described. The improvement procedures are presented in section 3. In section 4, the computational results are reported. Finally, some concluding remarks are provided in section 5.

2. The heuristic procedures

In this section, the considered heuristic procedures are described. First several dispatching rules are proposed. These rules include the forward scheduling EQTP procedure and several backward scheduling dispatching rules. Finally, a modified version of the MA_IN genetic algorithm is also considered.

Throughout this section, the focus will be on the problem version with inserted idle time. Therefore, the pseudo-codes and the algorithm descriptions provided are for this idle time version. These heuristic procedures can also be implemented for the problem with no idle time by removing all the procedures that insert idle time.

2.1 Dispatching rules

Dispatching rules were proposed and tested in Valente (2007b) for the problem without inserted idle time and the EQTP heuristic was found to be the best. In this section several new dispatching rules are proposed.

Since inserted idle time is allowed, a key consideration in developing the dispatching heuristics is how to incorporate inserted idle time. Schaller (2004) developed a procedure that determines the optimal amount of idle time to insert before each job in a given sequence of the jobs, i.e. the amount of idle time that minimizes the objective function value for that sequence. In this paper, a slightly modified version of this procedure is used. This modification consists only in a preprocessing phase that is performed prior to applying the procedure developed in Schaller (2004). In this preprocessing phase, the lateness of all jobs is first calculated assuming that no

idle time is used. Then, if all the jobs in the sequence are early, the minimum earliness value is calculated. Finally, this value is inserted as idle time before the first job. This preprocessing phase reduced considerably the computation time required by the timing procedure for the instances where most jobs are early. This modified version of the optimal timing procedure for the problem with inserted idle time will be denoted by OPT_IIT.

In the following, let S be the ordered set of the currently scheduled jobs and U be the set of the yet unscheduled jobs. Also, let $idle[i]$ represent the amount of idle time to be inserted before the ith position in the sequence, $i = 1, 2, \dots, n$. Finally, let LB_IIT denote the lower bounding procedure proposed by Schaller (2004) for the version with inserted idle time. This procedure, while calculating the lower bound, also computes values for the idle times before each position in the sequence. The pseudo-code for the dispatching heuristics for the version with inserted idle time is given in Algorithm 1.

Algorithm 1. Dispatching heuristics for the version with inserted idle time

1. Set $S = \varnothing$, $U = \{1, 2, \dots, n\}$.
2. Apply procedure LB_IIT to the instance and set $idle[i]$, $i = 1, 2, \dots, n$, equal to the corresponding idle time computed in procedure LB_IIT.
3. Use a dispatching rule to determine the sequence of jobs.
4. Apply procedure OPT_IIT to the sequence in set S and set $idle[i]$, $i = 1, 2, \dots, n$, equal to the corresponding optimal idle times computed in procedure OPT_IIT.
5. Calculate the objective function value of the schedule corresponding to the sequence in set S and the idle times in $idle[i]$, $i = 1, 2, \dots, n$.
6. Return sequence S, the idle times $idle[i]$, $i = 1, 2, \dots, n$ and the objective function value of the corresponding schedule.

In step 2, the amount of idle time before each position in the sequence is first set equal to the idle time values calculated in the lower bounding procedure LB_IIT. This approach proved greatly superior to another strategy that was considered in preliminary experiments, more specifically setting all idle times equal to 0. Step 3 uses one of the dispatching rules, to be described next, to generate a job sequence. In step 4, procedure OPT_IIT is used to obtain the optimal idle times for the sequence

generated in step 3. The objective function value of the corresponding schedule is calculated in step 5, and these elements are then returned in step 6.

2.1.1 The forward scheduling EQTP dispatching rule

The EQTP heuristic is the best of the existing dispatching rules for the problem with no idle time. The EQTP rule is a forward scheduling heuristic, i.e. the schedule is built from the front to the back, with jobs being added to the end of the current partial sequence. When the machine becomes available, the EQTP heuristic calculates a priority index for each unscheduled job, and the job with the highest priority value is then selected to be processed next. Let $I_j(t)$ denote the priority index of job j at the current time t (i.e. t is the time at which the next scheduled job will start). The EQTP heuristic uses the following priority index $I_j(t)$:

$$I_j(t) = \begin{cases} (1/p_j)\left[\bar{p} + 2(t + p_j - d_j)\right] & \text{if } s_j \le 0 \\ (\bar{p}/p_j)\exp\left[-(\bar{p}+1)s_j/k\bar{p}\right] & \text{if } 0 < s_j < \left[\bar{p}/(\bar{p}+1)k\bar{p}\right] \\ (1/p_j)^{-2}\left[(\bar{p}/p_j) - (1/p_j)(\bar{p}+1)s_j/k\bar{p}\right]^3 & \text{if } \left[\bar{p}/(\bar{p}+1)k\bar{p}\right] \le s_j < k\bar{p} \\ -(1/p_j) & \text{otherwise,} \end{cases}$$

where $s_j = d_j - t - p_j$ is the slack of job j, $\bar{p}$ is the average processing time of the remaining unscheduled jobs and k is a lookahead parameter.

The effectiveness of the EQTP rule depends on the value of the lookahead parameter k, which should reflect the number of competing critical jobs, i.e. the number of jobs that may clash each time a sequencing decision is to be made. The following procedure is used to calculate the value of k at each iteration. First, a critical slack value *crit_slack* is calculated. This critical slack value is calculated as *crit_slack* = *slack_prop* × $(n_U \times \bar{p} + \sum_{k=i+1}^{n} idle[k])$, where n_U is the number of unscheduled jobs, and $0 \le slack\_prop < 1$ is a user-defined parameter. Then, each job is classified as critical if $0 < s_j \le crit\_slack$, and non-critical otherwise. Therefore, a job is considered critical if it is not already tardy ($s_j > 0$), but is about to become tardy ($s_j \le crit\_slack$). Finally, the lookahead parameter k is set equal to the number of critical jobs. Note that in this procedure and other procedures proposed in this paper for the problem with inserted idle time, critical slack is defined as a proportion of the sum of

the remaining processing time and idle time. This differs from the definition for critical slack used in the problem without idle time, in which critical slack is defined as a proportion of the remaining processing time.

Dispatching Rule 1. EQTP dispatching rule for the version with inserted idle time

1. Set $t = 0$.
2. For $i = 1$ to n
   2.1. Set $t = t + idle[i]$.
   2.2. Calculate the EQTP priority index $I_j(t)$ for all jobs $j \in U$.
   2.3. Let l be the job with the largest priority index $I_j(t)$ for $j \in U$.
   2.4. Add job l to the end of set S and remove it from set U.
   2.5. Set $t = t + p_l$.

Table 1 provides the data for a numerical example that will be used to illustrate the proposed heuristics. When applied to the instance in example 1, the EQTP heuristic generates the sequence 3-5-4-2-1-6, with an objective function value of 3268. The details of the iterations performed by the EQTP procedure are provided in Table 2.

2.1.2. Backward scheduling dispatching rules

Preliminary experiments showed that the performance of the EQTP heuristic fell short of that desired, particularly for instances with relatively loose due dates, and therefore with a higher proportion of early jobs and larger amounts of inserted idle time. Comparisons with optimal sequences showed that the EQTP rule would sometimes erroneously select a job with a large processing time early in the construction of the sequence. This is due to the fact that at the beginning of the EQTP procedure, most or all jobs could be early, and the priority index would then lead to choosing a job with a large processing time, so as to reduce the earliness costs. However, this could cause a job to become tardy later in the sequencing process. Since the earliness costs are linear, while the tardiness costs are quadratic, and

therefore more heavily penalized, the initial reduction in the earliness costs would be more than offset by the later increase in the tardiness of one or more jobs.

Due to this particular nature of the objective function, which penalizes tardiness much more heavily, backward scheduling procedures were then considered. In backward scheduling heuristics, the schedule is built from the end to the front, with jobs being added to the beginning of the current partial sequence. With backward scheduling, the tardy jobs are considered first. Therefore, the backward heuristics consider right at the start the most important decisions, since they begin by scheduling the tardy jobs, which contribute the most to the objective function value.

In this subsection, four backward scheduling heuristics are presented. Two rules are first proposed that are standard in the sense that, at each iteration, they select the job with the largest priority. Then exchange checks are incorporated into these rules to create two additional rules. In these modified versions, the job with the largest priority value may not actually be the job selected for processing. Indeed, two types of exchange checks are performed, and these may lead to choosing a different job to be sequenced next.

The two backward scheduling dispatching rules without exchange checks are denoted by EQTP_Back and DR_Back. These rules share the same framework, and differ only in their priority index. The pseudo-code for these two backward scheduling rules without exchange checks is given next.

Dispatching Rules 2 and 3. Backward scheduling dispatching rules without exchange checks

1. Set $t = \sum_{j=1}^{n} p_j + \sum_{i=1}^{n} idle[i]$.
2. Apply procedure CT_Check.
3. For $i = n$ to 1
   - 3.1. Apply procedure Adjust_T_IIT to possibly adjust the current completion time t and the idle time before the previous positions.
   - 3.2. Calculate the priority index $I_j(t)$ for all jobs $j \in U$.
   - 3.3. Let l be the job with the largest priority index $I_j(t)$ for $j \in U$.
   - 3.4. Add job l to the beginning of set S and remove it from set U.
   - 3.5. Set $t = t - p_l - idle[i]$.

In step 1, the completion time t is initialized to the sum of the processing times of all jobs and the idle times calculated by procedure LB_IIT. In step 3.1, before calculating the priority of the unscheduled jobs, a procedure denoted by Adjust_T_IIT is applied. This procedure may modify the current completion time t, as well as the amount of idle time to be inserted before previous positions in the schedule. In order to do this, the Adjust_T_IIT procedure uses information previously obtained in step 2 by the application of procedure CT_Check.

The CT_Check and Adjust_T_IIT procedures were introduced after preliminary experiments showed that it was sometimes possible to detect that the existing inserted idle times were excessive. The two procedures were therefore developed to identify when LB_IIT inserted extra idle time, in order to then reduce the inserted idle times and the completion times. The introduction of these two procedures significantly improved the performance of the backward scheduling heuristics, particularly for instances with a higher proportion of early jobs.

The pseudo-code for the CT_Check and Adjust_T_IIT procedures, as well as two propositions that identify the presence of excessive inserted idle time (and therefore provide the theoretical basis for the two procedures), are presented in the appendix. The appendix also contains an example that is used to illustrate the CT_Check and Adjust_T_IIT procedures. As previously mentioned, the EQTP_Back and DR_Back heuristics differ only on the priority index. The EQTP_Back dispatching rule is basically a conversion of the EQTP heuristic to backward scheduling, and uses the following priority index $I_j(t)$:

$$I_j(t)=\begin{cases}(1/p_j) & \text{if } t-d_j\le 0\\ (1/p_j)\exp\left[-\left(1+1/\left(\bar{p}+2k\bar{p}\right)\right)\left(t-d_j\right)/k\bar{p}\right] & \text{if } 0<t-d_j<\left[1/\left(1+\bar{p}+2k\bar{p}\right)\right]k\bar{p}\\ \left[(1/p_j)\left(\bar{p}+2k\bar{p}\right)\right]^{-2}\left[(1/p_j)-(1/p_j)\left(1+\bar{p}+2k\bar{p}\right)\left(t-d_j\right)/k\bar{p}\right]^3 & \text{if } \left[1/\left(1+\bar{p}+2k\bar{p}\right)\right]k\bar{p}\le t-d_j<k\bar{p}\\ -(1/p_j)\left[\bar{p}+2\left(t-d_j\right)\right] & \text{otherwise}\end{cases}$$

As in the EQTP heuristic, the priority index of the EQTP_Back dispatching rule depends on the value of a lookahead parameter k. This parameter is again set equal to the number of critical jobs. However, a job j is now considered to be critical if $0 < t - d_j \le tardy\_prop \times t$, where $0 \le tardy\_prop < 1$ is a user-defined parameter.

The priority index of the DR_Back heuristic utilizes a value denoted by *min_tardy* which equals the minimum value of the tardiness among all jobs that are currently tardy. This priority index is given by the following expression:

$$I_j(t) = \begin{cases} (1/p_j) & \text{if } t - d_j \le 0 \\ -2(1/p'_j)(t - d_j) & \text{otherwise,} \end{cases}$$

where p'_j (a modified processing time) = $\min\{p_j, min\_tardy\}$. Both the EQTP_Back and DR_Back heuristics select an early job whenever such a job exists. This not only prevents the earliness of this job from increasing, but also decreases the tardiness of the unscheduled jobs that are currently still late. Whenever several early jobs exist, the expression of the priority index when $t - d_j \le 0$ assures that those jobs are scheduled in longest processing time order (when looking at the schedule from the front to the back), since this reduces the earliness costs of those jobs.

When all jobs are tardy (or quite tardy, in the case of the EQTP_Back heuristic), the priority index favours jobs with a low tardiness and a high processing time. Indeed, choosing a job with a low tardiness means that the increase in the objective function will be small. Also, selecting a job with a large processing time allows for a larger reduction in the tardiness of the remaining unscheduled jobs.

As previously mentioned, the reason for favouring jobs with larger processing times is the fact that this allows a larger decrease in the tardiness of the other jobs. However, if a job has a processing time that is equal to *min_tardy*, choosing this job now will allow the selection, at the next iteration, of a job that will then be right on-time. Giving more importance to jobs with processing times that are larger than *min_tardy* does not provide a benefit in this situation, since it would actually increase, in the next iteration, the earliness of the job that has currently the minimum tardiness *min_tardy*. Therefore, the original processing time was then replaced by the modified processing time p'_j in the second branch of the priority index of the DR_Back heuristic, which improved the results.

Table 3 and Table 4 illustrate the application of the EQTP_Back and DR_Back heuristics, respectively, to the instance in example 1. The EQTP_Back procedure generates the sequence 5-2-4-6-1-3, with an objective function value of

2544. The DR_Back procedure, on the other hand, obtains the sequence 3-5-6-1-4-2, with a cost of 2009.

The following two heuristics, denoted by EQTP_Back_Ex and DR_Back_Ex, are modified versions of the EQTP_Back and DR_Back heuristics, respectively. These modified versions were obtained by adding exchange checks to the original procedures. The exchange checks, as will be described below, may lead to choosing a job different from the one with the largest priority value.

Dispatching Rules 4 and 5. Backward scheduling dispatching rules with exchange checks

1. Set $t = \sum_{j=1}^{n} p_j + \sum_{i=1}^{n} idle[i]$.
2. Apply procedure CT_Check.
3. For $i = n$ to 1
   - 3.1. Apply procedure Adjust_T_IIT to possibly adjust the current completion time t and the idle time before the previous positions.
   - 3.2. Calculate the priority index $I_j(t)$ for all jobs $j \in U$.
   - 3.3. Let l be the job with the largest priority index $I_j(t)$ for $j \in U$.
   - 3.4. Apply procedure Exchange_Checks.
   - 3.5. Add job l to the beginning of set S and remove it from set U.
   - 3.6. Set $t = t - p_l - idle[i]$.

This procedure performs two sets of checks that may lead to selecting a job different from the one with the largest priority index $I_j(t)$. The pseudo-code for Exchange_Checks is given in Procedure 1.

Procedure 1. Exchange_Checks

1. Consider, in decreasing order of due dates, all jobs $h \in U \setminus \{l\}$ such that $d_k > d_l$:
   - 1.1. Let k be the job currently being considered.
   - 1.2. Calculate the cost $C_{k,l}$ of scheduling job l in position i and job k in position i - 1.

1.3. Calculate the cost $C_{l,k}$ of scheduling job k in position i and job l in position i - 1.

1.4. If $C_{l,k} < C_{k,l}$, set $l = k$ and move to step 2. Otherwise, consider the next job h.

2. Consider, in increasing order of processing times, all jobs $h \in U \setminus \{l\}$ such that $p_k < p_l$:

2.1. Let k be the job currently being considered.

2.2. Calculate the cost $C_{k,l}$ of scheduling job l in position i and job k in position i - 1.

2.3. Calculate the cost $C_{l,k}$ of scheduling job k in position i and job l in position i - 1.

2.4. If $C_{l,k} < C_{k,l}$, set $l = k$ and stop. Otherwise, consider the next job h.

Procedure Exchange_Checks performs two sets of checks that may change the job that will be chosen for processing at the current iteration. These checks were motivated by the several preliminary tests that were performed, as well as by the comparison of the EQTP_Back and DR_Back results with optimal sequences.

The first set of exchange checks is performed in step 1. In this step, the job currently selected is compared, in decreasing order of due dates, with the remaining unscheduled jobs that have a larger due date. For each candidate job the cost of that job and the currently selected job is then calculated when those jobs are scheduled in the next two positions, in the two possible orders. If the cost when the candidate job is scheduled at the current position is lower, that candidate job becomes the currently selected job, and the procedure then skips to step 2. Otherwise, the next candidate job is considered. The reason for including this set of checks is to avoid large squared tardiness penalties. Checking jobs with later due dates may reduce the squared tardiness enough to offset the benefits of scheduling the job with the largest priority index.

In step 2, the second set of exchange checks is performed. This second step is similar to the first, since it also calculates the cost of scheduling the currently selected job and the candidate job in the next two positions in the sequence, in the two possible orders. However, the candidate jobs are now the remaining unscheduled jobs that have a smaller processing time than the currently selected job. Furthermore, these candidate jobs are considered in increasing order of their processing times. The reason

for including this set of checks was that it was found that sometimes scheduling the job with the largest priority index (if it had a large processing time) caused the remaining jobs to be quite early. It was found that incorporating this set of checks sometimes scheduled a job with a lower processing time and a higher tardiness than the original job chosen, which resulted in a lower objective when the entire schedule was completed.

Again, example 1 is used to illustrate the application of the EQTP_Back_Ex and DR_Back_Ex heuristics. When applied to this instance, both of these procedures generate the sequence 3-5-4-1-6-2, with an objective function value of 1981. Although the two procedures give the same final sequence, they perform different exchange checks. The details of the iterations performed by the EQTP_Back_Ex and DR_Back_Ex heuristics are provided in Table 5 and Table 6, respectively.

2.2. Genetic algorithm

The MA_IN genetic algorithm is the best performing heuristic for the problem with no idle time. However, due to its computational requirements, this procedure can only be applied to small and medium size instances. In this procedure, a so-called initialization of the initial population is performed. This initialization consists in introducing in the initial population chromosomes that correspond to the sequences of four of the dispatching rules analysed in Valente (2007b). One of these four dispatching rules is precisely the EQTP heuristic that has been previously presented. Also, an adjacent pairwise interchange procedure is used to improve each chromosome in the successive populations. Therefore, the MA_IN procedure can be seen as a memetic algorithm, since it incorporates a local search procedure.

In this section, a modified version of this procedure, denoted as MA_IN_New, is proposed. The MA_IN_New heuristic is essentially identical to the MA_IN procedure, with the exception that, in the generation of the initial population, the chromosome corresponding to the solution of the EQTP heuristic has been replaced by a chromosome that contains the solution of the DR_Back_Ex heuristic. Indeed, the DR_Back_Ex procedure provided the best results among the dispatching rules that were considered, and its solution then replaced the EQTP sequence in the initial population of MA_IN_New. Furthermore, in the calculation of the fitness value of each chromosome, procedure OPT_IIT is also necessarily applied in order to

optimally insert idle time when calculating the objective function value of the sequence corresponding to that chromosome.

The MA_IN_New genetic algorithm was also applied to the version with no inserted idle time. For this version the procedure OPT_IIT is not required. However, and in addition to this usual difference between the procedures for the versions with and without idle time, there is one other distinction between the MA_IN_New algorithms for the two versions of the problem. This difference is in the way the local search procedure is applied. The discussion of this difference, however, will be deferred to the next section, where the improvement procedures are presented.

3. The improvement procedures

In this section, the proposed improvement procedures are described. Four improvement procedures have been considered. The first three are quite common in scheduling problems, and will only be briefly described. These three methods are the adjacent pairwise interchanges (API), 3-swaps (3SW) and first-improve interchanges (INTER) improvement procedures. The fourth procedure, even though it uses insertions, which are also common in scheduling problems, is more problem-specific. Therefore, this procedure, which will be denoted as INS, will be described in more detail.

The API procedure considers adjacent job positions. A pair of adjacent jobs is then swapped if such an interchange improves the objective function value. This process is repeated until no further improvement is found, i.e. until no additional improvement is possible through an adjacent swap.

The 3SW procedure is similar to the API method, but it considers three consecutive job positions, instead of only an adjacent pair of jobs. All possible permutations of these three jobs are then analysed, and the best configuration is selected. Once more, the procedure is applied repeatedly until no further improvement is possible.

The INTER procedure starts at the first position in the sequence, and then considers all the successive positions, until the end of the sequence is reached. For each position in the sequence, the INTER method considers interchanging the job that is currently scheduled in that position with the jobs that are scheduled in the following positions. Whenever such an interchange improves the objective function value, that

interchange is performed. If any interchange is performed before the end of the sequence is reached, the procedure starts again at the beginning of the schedule. Otherwise, no further improvement is possible, and the procedure terminates.

The application of improvement procedures to the version where idle time is allowed is made more complicated precisely due to that inserted idle time. Indeed, when an interchange is performed, there is a change in the sequence, and consequently the optimal amount of inserted idle time may therefore also change.

In Schaller (2004), an INTER improvement procedure had already been proposed. This method applied the OPT_IIT procedure to re-optimize the inserted idle times each time an interchange was performed. However, this made the procedure too computationally intensive, and therefore it could only be applied to small instances.

Applying the OPT_IIT procedure each time an interchange was performed would still make the improvement procedures too computationally demanding, and incapable of handling medium or large instances. Therefore, in this paper the API, 3SW and INTER procedures are first applied while keeping constant the idle times from the original schedule. Once no further improvement is possible, the OPT_IIT procedure is then used to re-optimize the inserted idle time. If the new idle times are different from the original ones, the improvement procedure is again applied (keeping the new idle times constant). This is repeated until there is no change in the idle times after the application of the improvement procedure. This approach was not only much more computationally efficient, but also performed well in terms of the improvement in the objective function value.

In the previous section, it was mentioned that there was a difference in the way the API local search procedure was applied in the MA_IN_New genetic algorithms for the two versions of the considered problem. For the version with no idle time, the API procedure is applied until no further improvement is possible. In the version with inserted idle time, however, only a single pass of the API procedure is performed. Therefore, in the MA_IN_New algorithm for the version with idle time the API procedure is not applied as described above. That is, while for the dispatching rules the API procedure is applied repeatedly until there is no change in the idle times, for the MA_IN_New genetic algorithm only a single pass of this procedure is performed. This is due to computational efficiency concerns. Indeed, the MA_IN_New algorithm applies the local search procedure to each chromosome. Employing only a single pass

of the API procedure allowed for a reduction in the computation time of the MA_IN_New algorithm, without compromising the solution quality.

The INS improvement procedure, as previously mentioned, is more problem-specific, and uses job insertions. In addition to insertions, this procedure also performs a rescheduling of the jobs between the insertion position and the original position of the job that was moved. The pseudo-code of the INS improvement method is given in procedure 2. In the following, let $index[i]$ denote the index of the job in position i, i = 1, 2, … , n. Also, let $pos(j)$ be the position of job j in the current schedule, j = 1, 2, … , n.

Procedure 2. INS improvement procedure

1. Determine the longest processing time (LPT) ordering of the jobs. Let LPT_k denote the index of the kth job, k = 1, 2, … , n, when the jobs are in LPT order (i.e. sequenced in non-increasing order of processing times).
2. For $j = LPT_1$ to LPT_n:
   - 2.1. Determine $pos(j)$. Set $ins\_pos = pos(j)$.
   - 2.2. For i = 1 to $pos(j)$ - 1:
     - 2.2.1. If $p_{index[i]} > p_j$, continue to the next position.
     - 2.2.2. If job $index[i]$ is not tardy if job j is inserted before it:
       - 2.2.2.1. Set $ins\_pos = i$ and go to step 2.3..
     - 2.2.3. Else:
       - 2.2.3.1. Calculate the cost $C_{index[i],j}$ of scheduling job $index[i]$ in position i and job j in position i + 1.
       - 2.2.3.2. Calculate the cost $C_{j,index[i]}$ of scheduling job j in position i and job $index[i]$ in position i + 1.
       - 2.2.3.3. If $C_{j,index[i]} < C_{index[i],j}$, set $ins\_pos = i$ and go to step 2.3..
   - 2.3. If $ins\_pos < pos(j)$:
     - 2.3.1. Calculate the cost $C_{current}$ of the current schedule.
     - 2.3.2. Create the following trial schedule from the current schedule:
       - 2.3.2.1. Move job j to position ins_pos.
       - 2.3.2.2. Reschedule the jobs in positions ins_pos + 1 to $pos(j)$ using the DR_Back_Ex rule.
     - 2.3.3. Calculate the cost C_{trial} of the trial schedule.

2.3.4. If $C_{trial} < C_{current}$, replace the current schedule with the trial schedule.

The INS improvement procedure was motivated by the comparison of the sequences generated by the dispatching rules with optimal sequences. Indeed, these comparisons showed that in some cases a job with a large processing time should have been scheduled earlier in the sequence. Therefore, in step 2, the INS procedure considers the jobs in non-increasing order of their processing times.

Let the job being currently considered for insertion earlier in the sequence be denoted as the insertion job. For each insertion job, its position in the sequence is first determined in step 2.1. Then, in step 2.2, a search is performed, in previous positions, for a candidate position for inserting the insertion job. In this search, and as can be seen by step 2.2.1, only positions that contain a job with a processing time that is not larger than the processing time of the insertion job are considered.

If one of those positions contains a job that will not be tardy even if the insertion job is inserted before it, then that position becomes the candidate position. Otherwise, the cost of the insertion job and the job in the current position is then calculated, when those two jobs are scheduled in the current and the next positions, in the two possible orders. If the cost when the insertion job is scheduled first is lower, then the current position becomes the candidate position.

If a candidate position is found, step 2.3 is then executed. In this step, the cost of the current schedule is first calculated. Then, a trial schedule is generated in step 2.3.2. This trial schedule is obtained by moving the insertion job to the candidate position, and rescheduling all the jobs between the next position and the original position of the insertion job using the DR_Back_Ex rule. The cost of this trial schedule is then calculated. If this cost is lower than that of the current schedule, the current schedule is replaced by the trial schedule.

Table 7 provides the data for an example that will be used to illustrate the INS procedure. The DR_Back_EX heuristic, when applied to the instance in example 2, will generate the sequence 3-6-1-2-4-5, with an objective function value of 3420. After the application of the INS improvement procedure, the sequence has been modified to 3-5-1-2-6-4, with a lower cost of 3384. The details of the iterations performed by the INS procedure are given in Table 8.

4. Computational results

In this section, the computational experiments and results are presented. First, the set of test problems used in the computational tests, and the parameter values that were selected for several heuristics, are described. Then, the computational results for the version with inserted idle time are presented. Finally, the results for the version where idle time is not allowed are analysed.

More extensive results are presented for the version with inserted idle time, since this is the main focus of this paper. For this version, the heuristics are first compared with the EQTP rule. Then, the effectiveness of the improvement procedures is analysed. Finally, the heuristic results are compared with optimum results. For the version with no idle time, only a summary of the main results is presented. Throughout this section, and in order to avoid excessively large tables, results will sometimes be presented only for some representative cases.

4.1. Experimental design and parameter values

The computational tests were performed on a set of problems with 10, 15, 20, 25, 30, 40, 50, 75, 100, 250, 500, 750 and 1000 jobs. These problems were randomly generated as follows. For each job j, an integer processing time p_j was generated from one of two uniform distributions [45, 55] and [1, 100], to create low (L) and high (H) variability, respectively. For each job j, an integer due date d_j is generated from the uniform distribution $[P(1 - T - R/2), P(1 - T + R/2)]$, where P is the sum of the processing times of all jobs, T is the tardiness factor and R is the range of due dates.

The range of due dates parameter was set at 0.2, 0.4, 0.6 and 0.8 for both the idle time and no idle time versions of the problem. The tardiness factor T was set at 0.0, 0.2, 0.4 and 0.6 for the version with inserted idle time. When idle time is not allowed, the additional values of 0.8 and 1.0 were also considered for this parameter. These two values were not used for the version with idle time, since for these high values of T most jobs are tardy, and the insertion of idle time does not improve the objective function.

For each combination of problem size n, processing time variability (var), T and R, 10 instances were randomly generated. Therefore, a total of 160 (240 for the problem without idle time) instances were generated for each combination of problem size and variability. The procedures for the version with inserted idle time were coded

in Turbo Pascal, and executed on a HP 1965 2.4 GHz personal computer. For the version with no idle time, all the algorithms were coded in Visual C++ 6.0, and executed on a Pentium IV 2.8 GHz personal computer.

The EQTP, EQTP_Back and EQTP_Back_Ex heuristics, require a value for the *slack_prop* and the *tardy_prop* parameters, respectively. For the EQTP heuristic on the problem with no idle time, a value of 0.60 was used, as recommended in Valente (2007b). Preliminary experiments were then performed to determine an appropriate value for *slack_prop* for the EQTP rule on the version with inserted idle time, as well as for the *tardy_prop* parameter for the EQTP_Back and EQTP_Back_Ex heuristics, for both versions of the problem.

The values {0.00, 0.05, 0.10, … ,0.95} were considered, and the objective function value was computed for each *slack_prop* or *tardy_prop* value and each instance. An analysis of these results then showed that *slack_prop* = 0.60 also proved adequate for the EQTP heuristic when idle time is allowed. For the EQTP_Back and EQTP_Back_Ex heuristics, and for both versions of the problem, the best results were obtained by setting *tardy_prop* at 0.05.

The MA_IN_New genetic algorithm requires values for several parameters. These parameters were set at the values that were previously used in Valente and Gonçalves (2008) for the MA_IN procedure for the version with no idle time.

4.2. Results for the version with inserted idle time

In this section, the computational results are presented for the version with inserted idle time. The heuristics are first compared with the EQTP rule. Then, the effectiveness of the improvement procedures is analysed. Finally, the heuristic results are compared with optimum results.

4.2.1. Comparison with the EQTP heuristic

In this section, the backward scheduling dispatching rules and the MA_IN_New genetic algorithm are compared with the forward scheduling EQTP heuristic. Table 9 provides the mean relative improvement in objective function value over the EQTP rule. The relative improvement is calculated as (eqtp_ofv – heur_ofv) /

eqtp_ofv × 100, where eqtp_ofv and heur_ofv are the objective function values of the EQTP rule and the appropriate heuristic, respectively.

The backward scheduling rules and the MA_IN_New genetic algorithm clearly outperform the EQTP heuristic. The best performance, as expected, is provided by the MA_IN_New genetic procedure. However, the best performing of the backward dispatching heuristics gives results that are extremely close to those of the genetic algorithm.

The performance of the backward scheduling rules is somewhat similar. However, slightly better results are obtained when the DR priority index and / or the exchange checks procedure are used. In fact, the DR_Back(_Ex) heuristic is usually marginally superior to its EQTP_Back(_Ex) counterpart. Also, the EQTP_Back_Ex and DR_Back_Ex procedures are superior to the respective versions that do not incorporate the exchange checks.

The processing time variability has a significant impact on the improvement provided by the heuristic procedures. As can be seen by the results in Table 9, the improvement over the EQTP rule is much higher for the instances with high variability. Indeed, the relative improvement is about 4% when the variability is low, but it usually ranges from 15% to 19% for high variability instances.

Table 10 gives the effect of the T and R parameters on the relative improvement over the EQTP rule, for instances with 100 jobs. The heuristic results are quite close when $T = 0.6$. For these instances, there are a larger proportion of tardy jobs, and little or no inserted idle time is required. The relative improvement is also quite minor when $T = 0.4$ and the due dates are not widely spread. However, the relative improvement is quite high (larger than 40% for some parameter combinations) for the lower values of the tardiness factor (as well as for the $T = 0.4$ and $R = 0.8$ parameter combination). Therefore, a larger improvement is obtained for instances with a higher proportion of early jobs ($T = 0.0$ or 0.2), where larger amounts of inserted idle time are needed.

The heuristic runtimes (in seconds) are provided in Table 11. The MA_IN_New genetic algorithm is computationally quite demanding, and can only be applied to small and medium size instances. The dispatching rules, however, are very efficient, and can quickly solve even quite large problems. The versions that incorporate the exchange checks do require a little additional time when compared

with their EQTP_Back and DR_Back counterparts, but are nevertheless still quite efficient.

The DR_Back_Ex dispatching rule then seems to be the heuristic procedure of choice. Indeed, its performance in terms of solution quality is virtually identical to that of the MA_IN_New genetic algorithm, which provides the best results. Also, the DR_Back_Ex rule is computationally quite efficient, and can quickly solve even quite large instances.

4.2.2. The improvement procedures

In this section, the computational results for the improvement procedures are presented. As mentioned in the previous section, the DR_Back_Ex dispatching rule is the most effective of the considered heuristics, since it not only performs well in terms of solution quality, but is also computationally quite efficient. Therefore, in this section the improvement procedures are only applied to the DR_Back_Ex dispatching rule, in order to see if it is possible to further improve the schedules generated by this heuristic. Due to the large computational times that would be required, the improvement procedures were only applied on instances with up to 500 jobs.

Table 12 provides the mean relative improvement in objective function value provided by the improvement procedures. The improvement procedures only provide a minor improvement. Nevertheless, the processing time variability has a major impact on the effect of the improvement procedures. Indeed, when the variability is low, the improvement procedures are nearly ineffective. For instances with high variability, however, the relative improvement increases significantly, although it is still usually less than 0.5%. The INTER and INS procedures usually provide a larger improvement than the API and 3SW methods. Also, the relative improvement given by the 3SW procedure is higher than that provided by the API method.

Table 13 gives the effect of the T and R parameters on the relative improvement provided by the improvement procedures, for instances with 100 jobs. The relative improvement is quite minor for instances with $T = 0.4$ or 0.6, where little or no inserted idle time is required. The only exception occurs when the due dates are quite close ($R = 0.2$), since in this case the INTER and INS procedures do provide a larger improvement. The relative improvement is visibly higher for the instances with

a higher proportion of early jobs (T = 0.0 or 0.2), where larger amounts of inserted idle time are needed.

The runtimes required by the improvement procedures (in seconds) are provided in Table 14. The INTER procedure is computationally quite intensive, and can therefore only be applied to small or medium size instances. The INS procedure is also somewhat demanding, but it is much faster than the INTER procedure, and can be applied to large instances. The 3SW and API procedures are quite fast, and can then be used even for quite large problems. The INS and INTER procedures are then recommended for small and medium size instances, while the INS method can be used even for large problems. For extremely large instances, the 3SW procedure can still be applied efficiently.

4.2.3. Comparison with optimum results

In this section, the heuristic results are compared with the optimum objective function values, for instances with up to 20 jobs. Table 15 gives the mean relative deviation from the optimum (%dev), calculated as (H – O) / O × 100, where H and O are the heuristic and the optimum objective function values, respectively. The percentage number of times each heuristic generates an optimum schedule (%opt) is also provided. In the following, let DR_Back_Ex + INS denote the DR_Back_Ex rule, followed by the application of the INS improvement procedure.

The MA_IN_New and DR_Back_Ex + INS heuristics perform extremely well. Indeed, the relative deviation from the optimum is quite small for these heuristics. Also, these procedures provide an optimum solution for a quite large number of the test instances.

The DR_Back_Ex and EQTP_Back_Ex heuristics also provide quite good results. As previously mentioned, the incorporation of the exchange checks improved the heuristic results. This can also be clearly seen in Table 15, since the DR_Back_Ex and EQTP_Back_Ex procedures outperform their counterparts that do not perform the exchange checks. Nevertheless, even the DR_Back and EQTP_Back heuristics noticeably outperform the forward scheduling EQTP rule.

From Table 15, it can be seen that the heuristics are much closer to the optimum when the processing time variability is low. This is quite clear for the worst performing heuristics, particularly for the EQTP rule. Indeed, for the low variability

instances, the EQTP heuristic is 3-7% above the optimum. When the variability is high, however, the deviation from the optimum is over 25%.

This is in line with the results previously presented in Table 9 and Table 12. Indeed, the improvement over the EQTP rule provided by the heuristic procedures was much higher for the instances with high variability. Similarly, the improvement procedures provided a larger improvement when the processing time variability was high. This is in accordance with the results given in Table 15, since there is indeed more room for improvement when the variability is high.

Table 16 gives the effect of the T and R parameters on the relative deviation from the optimum, for instances with 20 jobs. The relative deviation from the optimum is usually higher for instances with a low tardiness factor, where larger amounts of inserted idle time are required, particularly for the worst performing heuristics. Again, this is in accordance with the results presented in Table 10 and Table 13. In fact, both the improvement over the EQTP rule and the improvement provided by the improvement procedures were higher for instances with $T = 0.0$ or 0.2.

4.3. Results for the version with no idle time

In this section, a summary of the main computational results is presented for the version with no idle time. Table 17 provides the mean relative improvement in objective function value over the EQTP rule. The results in Table 17 are similar to those previously presented for the version with inserted idle time.

Indeed, the backward scheduling rules and the genetic algorithms clearly outperform the EQTP heuristic. The best performance is again provided by the genetic algorithms and the DR_Back_Ex rule. Moreover, the use of the DR priority index and / or the exchange checks procedure lead to slightly better results. The processing time variability also once more has a significant effect on the improvement given by the heuristic procedures. In fact, the improvement over the EQTP rule is again much higher for the high variability instances.

Table 18 provides the mean relative deviation from the optimum (%dev), as well as the percentage number of times each heuristic generates an optimum schedule (%opt). Once more, the results in Table 18 are similar to those previously given for the inserted idle time version. The genetic algorithms and the DR_Back_Ex + INS

procedure perform extremely well, giving a relative deviation from the optimum that is inferior to 0.1% and providing an optimum solution for nearly all of the instances.

The DR_Back_Ex and EQTP_Back_Ex heuristics also provide excellent results. It can again be seen that the incorporation of the exchange checks procedures improves the heuristic performance. Indeed, the DR_Back_Ex and EQTP_Back_Ex procedures provide better results than their DR_Back and EQTP_Back counterparts. Nevertheless, even these latter two heuristics clearly outperform the EQTP rule. Once more, the heuristics, particularly the ones that perform worse, are also closer to the optimum when the processing time variability is low.

The DR_Back_Ex dispatching rule, followed by an improvement procedure such as the INS method, then again seems to be the heuristic procedure of choice. Indeed, and on the one hand, its performance in terms of solution quality is virtually identical to that of the MA_IN_New genetic algorithm, which provides the best results. On the other hand, the DR_Back_Ex + INS procedure is computationally quite efficient, and can quickly solve even large instances.

5. Conclusion

This paper considered the single machine scheduling problem with linear earliness and quadratic tardiness costs. The problem is of practical relevance when idle time may be inserted, as well as when the insertion of idle time is not allowed. Past research on the version with inserted idle time focused on an exact approach, while several heuristics have already been proposed for the version with no idle time. Among these heuristics, the MA_IN genetic algorithm provides the best results for small and medium size instances, while the EQTP dispatching rule is the best procedure for large instances.

These algorithms were the basis for the development of some new heuristic procedures for the version with inserted idle time. In this context, several backward scheduling dispatching rules were proposed, as well as a slightly modified genetic algorithm. Also, some improvement procedures were considered, in order to improve the schedule generated by the heuristics.

The new algorithms significantly outperformed the forward scheduling EQTP rule. Therefore, a backward scheduling approach proved to be superior for the considered problem, which is likely due to the fact that the tardiness costs are more

heavily penalized in the objective function. The dispatching rules that included an exchange check procedure also performed better than their counterparts that did not include such a procedure.

The MA_IN_New genetic algorithm provides the best results in terms of solution quality, but is computationally quite intensive, and can only be applied to small and medium size instances. The DR_Back_Ex dispatching rule, followed by an improvement procedure such as the INS method, emerges as the most effective heuristic. In fact, not only its solution quality is virtually identical to that of the MA_IN_New genetic algorithm, but this heuristic is also computationally quite efficient, and can quickly solve even large instances.

The new heuristics were also applied, with the appropriate modifications, to the version with no idle time. The computational results were similar to those obtained for the version with inserted idle time. Therefore, the procedures developed in this paper are the new heuristics of choice for both versions of the single machine scheduling problem with linear earliness and quadratic tardiness costs.

Acknowledgement

The authors would like to thank the anonymous referees for several comments and suggestions that have been used to improve this paper.

Appendix

As previously mentioned in section 2.1.2., the CT_Check and Adjust_T_IIT procedures were introduced to identify when LB_IIT inserted extra idle time, in order to then reduce the inserted idle times and the completion times. In this appendix, two propositions that provide the theoretical foundation for these two procedures, since they are used to identify the presence of excessive inserted idle times, are first presented. Then, the pseudo-code for the CT_Check and Adjust_T_IIT procedures is given.

Let EDD_k denote the index of the kth job, $k = 1, 2, \ldots, n$, when the jobs are in earliest due date order (i.e. sequenced in non-decreasing order of due dates).

Similarly, let SPT_k denote the index of the kth job, $k = 1, 2, \ldots, n$, when the jobs are in shortest processing time order (i.e. sequenced in non-decreasing order of processing times). Also, please recall that $idle[i]$ and $index[i]$, $i = 1, 2, \ldots, n$, represent the amount of idle time before position i and the index of the job in position i, respectively. Finally, let $I_k = \sum_{i=1}^{k} idle[i]$.

Proposition 1. In an optimal solution, if $d_{index[k]} + (k-1)/2 \geq \sum_{i=1}^{k} p_{index[i]}$, then $I_k \leq d_{index[k]} + (k-1)/2 - \sum_{i=1}^{k} p_{index[i]}$; otherwise, $I_k = 0$.

Proof. Let $I_k = \max\{d_{index[k]} + (k-1)/2 - \sum_{i=1}^{k} p_{index[i]}, 0\}$, which implies $T_{index[k]} \geq (k-1)/2$. Let $I'_k = I_k + \Delta I_k$, for $\Delta I_k \geq 0$. If the inserted idle times result in I'_k instead of I_k, then the total earliness of the first $k-1$ jobs in the sequence will decrease at most $\Delta I_k(k-1)$. The squared tardiness of the job in position k will increase by $(T_{index[k]} + \Delta I_k)^2 - T_{index[k]}^2 = 2\Delta I_k T_{index[k]} + \Delta I_k^2$. The objective function value will not increase if $2\Delta I_k T_{index[k]} + \Delta I_k^2 \leq \Delta I_k(k-1)$. Since $T_{index[k]} \geq (k-1)/2$, $2\Delta I_k T_{index[k]} + \Delta I_k^2 \leq \Delta I_k(k-1)$ holds only if $\Delta I_k = 0$.■

Proposition 2. Assume that $p_j \geq 1$, $j = 1, 2, \ldots, n$, and let $l = EDD_k$. Then, in an optimal solution, if $d_l + (k-1)/2 \geq \sum_{i=1}^{k} p_{index[i]}$, then $I_k \leq d_l + (k-1)/2 - \sum_{i=1}^{k} p_{index[i]}$; otherwise, $I_k = 0$.

Proof. To prove this proposition, three cases are considered: 1) $index[k] = l = EDD_k$; 2) $index[k] = EDD_j, j < k$ and 3) $index[k] = EDD_j, j > k$.

Case 1: $index[k] = l = EDD_k$. The proof for this case is the same as the proof for Proposition 1, and is therefore omitted.

Case 2: $index[k] = EDD_j, j < k$. In this case, $d_{index[k]} \leq d_l$. By Proposition 1, if $d_{index[k]} + (k-1)/2 \geq \sum_{i=1}^{k} p_{index[i]}$, then $I_k \leq d_{index[k]} + (k-1)/2 - \sum_{i=1}^{k} p_{index[i]}$, which implies that $I_k \leq d_l + (k-1)/2 - \sum_{i=1}^{k} p_{index[i]}$. Also by Proposition 1, if $d_{index[k]} + (k-1)/2 < \sum_{i=1}^{k} p_{index[i]}$, then $I_k = 0$, which implies that $I_k \leq d_l + (k-1)/2 - \sum_{i=1}^{k} p_{index[i]}$ if $d_l + (k-1)/2 \geq \sum_{i=1}^{k} p_{index[i]}$.

Case 3: $index[k] = EDD_j$, $j > k$. In this case, there is a job in position m of the sequence such that $m > k$ and $d_{index[m]} \leq d_l$. Note that $I_k \leq I_m$. By Proposition 1, if $d_{index[m]} + (m - 1)/2 < \sum_{i=1}^{m} p_{index[i]}$, then $I_m = 0$, which implies that $I_k = 0$. Also by Proposition 1, if $d_{index[m]} + (m - 1)/2 \geq \sum_{i=1}^{m} p_{index[i]}$, then $I_m \leq d_{index[m]} + (m - 1)/2 - \sum_{i=1}^{m} p_{index[i]} = d_{index[m]} + (m - k)/2 + (k - 1)/2 - \sum_{i=1}^{k} p_{index[i]} - \sum_{i=k+1}^{m} p_{index[i]}$. Since $p_j \geq 1, j = 1, 2, \ldots, n$, then $\sum_{i=k+1}^{m} p_{index[i]} > (m - k)/2$. Therefore, $I_m \leq d_{index[m]} + (k - 1)/2 - \sum_{i=1}^{k} p_{index[i]} \leq d_l + (k - 1)/2 - \sum_{i=1}^{k} p_{index[i]}$, which implies that $I_k \leq d_l + (k - 1)/2 - \sum_{i=1}^{k} p_{index[i]}$.■

From proposition 2, it follows directly that if $p_j \geq 1, j = 1, 2, \ldots, n$, then in an optimal solution $C_{index[k]} \leq \max \{d_l + (k - 1)/2, \sum_{i=1}^{k} p_{index[i]}\}$, for $k = 1, 2, \ldots, n$. The pseudo-code for the CT_Check and Adjust_T_IIT procedures can now be presented. Procedure CT_Check first sets an initial completion time for each position i of the sequence ($i = 1, 2, \ldots, n$), denoted by CT_C_i. Procedure Adjust_T_IIT then reduces the current completion time t to the maximum of CT_C_i and the remaining processing time. If t is reduced, then the cumulative idle time is also reduced.

Procedure 3. CT_Check

1. Set $P = \sum_{j=1}^{n} p_j$.
2. For $i = n$ to 1
   - 2.1. Set $k = EDD_i$.
   - 2.2. Set $l = SPT_i$.
   - 2.3. Set $maxt = (i - 1)/2$ if i is even and $\lfloor (i-1)/2 \rfloor + 1$ if i is odd, where $\lfloor\ \rfloor$ is the integer portion of a real number.
   - 2.4. Set $CT\_C_i = \max \{d_k + maxt, P\}$.
   - 2.5. Set $P = P - p_l$.

Procedure 4. Adjust_T_IIT

1. Set $P = \sum_{j \in U} p_j$.
2. If $t > \max\{CT\_C_i, P\}$, then set $reduct = t - \max\{CT\_C_i, P\}$; otherwise, set $reduct = 0$.
3. If $reduct > 0$, then set $t = t - reduct$.
4. Set $j = i$.
5. While $reduct > 0$
   - 5.1. If $reduct \geq idle[j]$:
     - 5.1.1. Set $reduct = reduct - idle[j]$.
     - 5.1.2. Set $idle[j] = 0$.
   - 5.2. Else:
     - 5.2.1. Set $idle[j] = idle[j] - reduct$.
     - 5.2.2. Set $reduct = 0$.
   - 5.3. If $j > 0$, set $j = j - 1$; otherwise set $reduct = 0$.

Table 19 contains an example that is used to illustrate the CT_Check and Adjust_T_IIT procedures. The details of the application of these procedures to the instance in example 3 are provided in Table 20.

References

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

Hoogeveen, H. (2005) 'Multicriteria scheduling', *European Journal of Operational Research*, Vol. 167, pp.592–623.

Kanet, J.J. and Sridharan, V. (2000) 'Scheduling with inserted idle time: problem taxonomy and literature review', *Operations Research*, Vol. 48, pp.99–110.

Schaller, J. (2004) 'Single machine scheduling with early and quadratic tardy penalties', *Computers and Industrial Engineering*, Vol. 46, pp.511–532.

Taguchi, G. (1986) *Introduction to Quality Engineering*, Asian Productivity Organization, Tokyo, Japan.

Valente, J.M.S. (2007a) *Beam Search Heuristics for the Single Machine Scheduling Problem with Linear Earliness and Quadratic Tardiness Costs*, Working Paper 250, Faculdade de Economia, Universidade do Porto, Portugal (to appear in *Asia-Pacific Journal of Operational Research*).

Valente, J.M.S. (2007b) 'Heuristics for the single machine scheduling problem with early and quadratic tardy penalties', *European Journal of Industrial Engineering*, Vol. 1, pp.431-448.

Valente, J.M.S. (2008) 'An exact approach for the single machine scheduling problem with linear early and quadratic tardy penalties', *Asia-Pacific Journal of Operational Research*, Vol. 25, pp.169-186.

Valente, J.M.S. and Gonçalves, J.F. (2008) *A Genetic Algorithm Approach For The Single Machine Scheduling Problem With Linear Earliness And Quadratic Tardiness Penalties*, Working Paper 264, Faculdade de Economia, Universidade do Porto, Portugal.

Wagner, B.J., Davis, D.J. and Kher, H. (2002) 'The production of several items in a single facility with linearly changing demand rates', *Decision Sciences*, Vol. 33, pp.317–346.

Table 1 Example 1 data

job (j)	1	2	3	4	5	6
p_j	10	40	72	39	52	25
d_j	180	196	191	194	168	194

Table 2 EQTP example

							s_j						$I_j(t)$						chosen
iter	t	$\bar{p}$	cut_1	cut_2	k	*c_slk*	1	2	3	4	5	6	1	2	3	4	5	6	job
1	0	39.67	77.38	79.33	2	142.80	170	156	119	155	116	169	-0.10000	-0.02500	-0.01390	-0.02560	-0.01900	-0.04000	3
2	72	33.20	161.15	166.00	5	99.60	98	84	NA	83	44	97	0.00000	0.00000	NA	0.00000	0.00007	0.00000	5
3	124	28.50	110.14	114.00	4	68.40	46	32	NA	31	NA	45	0.00002	0.00018	NA	0.00024	NA	0.00001	4
4	163	25.00	48.08	50.00	2	45.00	7	-7	NA	NA	NA	6	0.06500	0.97500	NA	NA	NA	0.04400	2
5	203	17.50	0.00	0.00	0	21.00	-33	NA	NA	NA	NA	-34	8.35000	NA	NA	NA	NA	3.42000	1
6	213	25.00	0.00	0.00	0	15.00	NA	NA	NA	NA	NA	-44	NA	NA	NA	NA	NA	4.52000	6

(a) initial and final idle time is 0 for all positions
(b) *slack_prop* = 0.60
(c) cut_1 = $\left\lceil \bar{p} / (\bar{p}+1) k\bar{p} \right\rceil$, cut_2 = $k\bar{p}$ and *c_slk* = *crit_slack*

Table 3 EQTP_Back example

							$t - d_j$						$I_j(t)$						chosen
iter	t	$\bar{p}$	cut_1	cut_2	k	*c_slk*	1	2	3	4	5	6	1	2	3	4	5	6	job
1	238	39.67	0.00	0.00	0	11.90	58	42	47	44	70	44	-15.570	-3.090	-1.860	-3.270	-3.460	-5.110	3
2	166	33.20	0.00	0.00	0	8.30	-14	-30	NA	-28	-2	-28	0.100	0.025	NA	0.026	0.019	0.040	1
3	156	39.00	0.00	0.00	0	7.80	NA	-40	NA	-38	-12	-38	NA	0.025	NA	0.026	0.019	0.040	6
4	131	43.67	0.00	0.00	0	6.55	NA	-65	NA	-63	-37	NA	NA	0.975	NA	0.026	0.019	NA	4
5	92	46.00	0.00	0.00	0	4.60	NA	-104	NA	NA	-76	NA	NA	0.975	NA	NA	0.019	NA	2
6	52	52.00	0.00	0.00	0	2.60	NA	NA	NA	NA	-116	NA	NA	NA	NA	NA	0.019	NA	5

(a) initial and final idle time is 0 for all positions
(b) *tardy_prop* = 0.05
(c) cut_1 = $\left\lceil 1/\left(1+\bar{p}+2k\bar{p}\right) \right\rceil k\bar{p}$, cut_2 = $k\bar{p}$ and *c_slk* = *tardy_prop* * *t*

Table 4 DR_Back example

			$t - d_j$						$I_j(t)$						chosen
iter	t	*min_tardy*	1	2	3	4	5	6	1	2	3	4	5	6	job
1	238	42	58	42	47	44	70	44	-11.6000	-2.1000	-2.2400	-2.2600	-3.3300	-3.5200	2
2	198	4	18	NA	7	4	30	4	-9.0000	NA	-3.5000	-2.0000	-15.0000	-2.0000	4
3	159	NA	-21	NA	-32	NA	-5	-6	0.1000	NA	0.0139	NA	0.0192	0.0400	1
4	149	NA	NA	NA	-42	NA	-19	-45	NA	NA	0.0139	NA	0.0192	0.0400	6
5	124	NA	NA	NA	-67	NA	-44	NA	NA	NA	0.0139	NA	0.0192	NA	5
6	72	NA	NA	NA	-119	NA	NA	NA	NA	NA	0.0139	NA	NA	NA	3

(a) initial and final idle time is 0 for all positions

Table 5 EQTP_Back_Ex example

iter	t	l_j	type	cur	cand	ord_1	ofv_1	ord_2	ofv_2	sel	F_j
1	238	3									2
			1	3	2	3-2	1813	2-3	2239	2	
			2	2	1	2-1	4388	1-2	2088	2	
			2	2	6	2-6	2225	6-2	1780	2	
			2	2	4	2-4	1945	4-2	1780	2	
2	198	4									6
			2	4	1	4-1	330	1-4	37	4	
			2	4	6	4-6	37	6-4	51	6	
3	173	1									1
			1	1	4	1-4	67	4-1	38	1	
			1	1	3	1-3	97	3-1	35	1	
4	163	4									4
5	124	5									5
			1	5	3	5-3	183	3-5	163	5	
6	72	3									3

(a) initial and final idle time is 0 for all positions
(b) l_j is the job selected before the exchange checks, type is the type of exchange check, cur is the currently selected job, cand is the candidate job, ord_1 is the order of the two jobs with the current job placed first and ofv_1 is the cost of these two jobs in this order, ord_2 is the order of the two jobs with the candidate job placed first and ofv_2 is the cost of these two jobs in this order, sel is the job selected to be placed last among the current job and the candidate job and F_j is the final job that was selected after the exchange checks

Table 6 DR_Back_Ex example

iter	t	l_j	type	cur	cand	ord_1	ofv_1	ord_2	ofv_2	sel	F_j
6	238	2									2
			2	2	1	2-1	4388	1-2	2088	2	
			2	2	6	2-6	2225	6-2	1780	2	
			2	2	4	2-4	1945	4-2	1780	2	
5	198	4									6
			2	4	1	4-1	330	1-4	37	4	
			2	4	6	4-6	37	6-4	51	6	
4	173	1									1
3	163	4									4
2	124	5									5
1	72	3									3

(a) initial and final idle time is 0 for all positions
(b) l_j is the job selected before the exchange checks, type is the type of exchange check, cur is the currently selected job, cand is the candidate job, ord_1 is the order of the two jobs with the current job placed first and ofv_1 is the cost of these two jobs in this order, ord_2 is the order of the two jobs with the candidate job placed first and ofv_2 is the cost of these two jobs in this order, sel is the job selected to be placed last among the current job and the candidate job and F_j is the final job that was selected after the exchange checks

Table 7 Example 2 data

job (j)	1	2	3	4	5	6
p_j	36	16	79	77	59	41
d_j	219	216	248	253	253	235

Table 8 INS procedure example

k	*j*	*pos*(*j*)	*ins_pos*	$C_{current}$	trial schedule	C_{trial}	$C_{trial} < C_{current}$
1	3	1					
2	4	5	2	3420	3-4-1-2-6-5	3522	No
3	5	6	2	3420	3-5-1-2-6-4	3384	Yes
4	6	5	3	3384	3-5-6-2-1-4	3530	No
5	1	3					

Table 9 Relative improvement over the EQTP heuristic

var	*n*	EQTP_Back	DR_Back	EQTP_Back_Ex	DR_Back_Ex	MA_IN_New
L	15	4.75	4.82	4.85	4.88	4.90
	25	3.95	4.01	4.02	4.07	4.09
	50	4.09	4.10	4.14	4.14	4.15
	100	3.67	3.68	3.69	3.69	3.70
	250	3.91	3.91	3.91	3.91	---
	500	4.12	4.12	4.13	4.13	---
	1000	4.14	4.14	4.14	4.14	---
H	15	6.92	12.90	14.64	15.17	15.52
	25	11.83	13.20	15.05	15.39	15.63
	50	14.52	14.81	15.30	15.36	15.62
	100	16.72	16.51	16.98	16.98	17.25
	250	18.31	18.24	18.37	18.38	---
	500	18.38	18.41	18.45	18.46	---
	1000	18.93	19.00	19.26	19.01	---

Table 10 Relative improvement over the EQTP heuristic for instances with 100 jobs

		low var				high var			
heur	*T*	*R*=0.2	*R*=0.4	*R*=0.6	*R*=0.8	*R*=0.2	*R*=0.4	*R*=0.6	*R*=0.8
EQTP_Back	0.0	1.56	1.59	5.51	6.58	4.42	18.73	33.86	42.35
	0.2	22.16	3.86	5.36	6.85	42.02	23.04	31.26	41.37
	0.4	0.64	0.01	0.12	4.51	0.79	0.06	0.97	29.38
	0.6	-0.01	0.00	0.00	-0.01	0.09	-0.11	-0.16	-0.42
DR_Back	0.0	1.57	1.59	5.51	6.58	4.46	18.73	33.79	42.34
	0.2	22.24	3.86	5.36	6.85	41.78	22.58	31.26	40.59
	0.4	0.64	0.02	0.14	4.52	0.90	0.20	1.21	26.26
	0.6	0.00	0.00	0.00	0.00	0.19	-0.02	-0.07	-0.10
EQTP_Back_Ex	0.0	1.59	1.62	5.54	6.58	4.44	18.60	33.94	42.92
	0.2	22.27	3.86	5.37	6.86	42.55	23.04	31.36	42.18
	0.4	0.65	0.03	0.15	4.52	1.08	0.33	1.55	29.09
	0.6	0.00	0.00	0.00	0.00	0.35	0.07	0.06	0.10
DR_Back_Ex	0.0	1.59	1.62	5.54	6.58	4.46	18.60	33.74	42.48
	0.2	2.27	3.86	5.37	6.86	42.65	23.08	31.36	41.67
	0.4	0.65	0.03	0.15	4.53	1.09	0.34	1.67	29.88
	0.6	0.00	0.00	0.00	0.00	0.35	0.07	0.06	0.10
MA_IN_New	0.0	1.59	1.62	5.54	6.62	4.63	18.88	34.32	43.05
	0.2	22.27	3.87	5.37	6.88	42.96	23.13	31.47	43.13
	0.4	0.65	0.03	0.15	4.55	1.30	0.51	1.74	30.05
	0.6	0.00	0.00	0.00	0.00	0.52	0.11	0.07	0.13

Table 11 Heuristic runtimes (in seconds)

var	*n*	EQTP	EQTP_Back	DR_Back	EQTP_Back_Ex	DR_Back_Ex	MA_IN_New
L	25	0.00	0.01	0.00	0.00	0.00	1.21
	50	0.01	0.02	0.01	0.01	0.01	6.99
	100	0.05	0.06	0.04	0.05	0.05	64.23
	250	0.21	0.19	0.20	0.22	0.20	---
	500	0.63	0.67	0.57	0.73	0.62	---
	1000	2.73	2.77	2.25	3.08	2.56	---
H	25	0.00	0.01	0.00	0.00	0.00	0.69
	50	0.01	0.02	0.01	0.01	0.01	9.67
	100	0.05	0.06	0.05	0.03	0.04	75.64
	250	0.20	0.19	0.19	0.22	0.21	---
	500	0.61	0.69	0.58	0.73	0.63	---
	1000	2.50	2.75	2.20	8.61	2.65	---

Table 12 Improvement procedures - relative improvement

var	*n*	API	3SW	INTER	INS
L	15	0.00	0.00	0.03	0.03
	25	0.00	0.00	0.01	0.02
	50	0.01	0.01	0.02	0.02
	100	0.01	0.01	0.02	0.02
	250	0.00	0.01	0.02	0.01
	500	0.00	0.01	0.02	0.01
H	15	0.02	0.07	0.20	0.52
	25	0.08	0.11	0.19	0.20
	50	0.13	0.22	0.38	0.21
	100	0.24	0.36	0.51	0.31
	250	0.12	0.17	0.30	0.17
	500	0.11	0.15	0.27	0.17

Table 13 Improvement procedures - relative improvement for instances with 100 jobs

		low var				high var			
imp proc	*T*	*R*=0.2	*R*=0.4	*R*=0.6	*R*=0.8	*R*=0.2	*R*=0.4	*R*=0.6	*R*=0.8
API	0.0	0.00	0.00	0.00	0.04	0.14	0.34	0.60	0.68
	0.2	0.00	0.00	0.01	0.02	0.04	0.02	0.17	1.82
	0.4	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.07
	0.6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01
3SW	0.0	0.00	0.00	0.00	0.06	0.15	0.42	0.89	1.13
	0.2	0.00	0.00	0.01	0.03	0.05	0.06	0.31	2.44
	0.4	0.00	0.00	0.00	0.01	0.00	0.03	0.07	0.13
	0.6	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.02
INTER	0.0	0.02	0.00	0.01	0.10	0.22	0.55	1.14	1.39
	0.2	0.00	0.01	0.02	0.10	0.47	0.06	0.45	3.04
	0.4	0.00	0.00	0.00	0.04	0.18	0.16	0.07	0.16
	0.6	0.00	0.00	0.00	0.00	0.14	0.03	0.01	0.03
INS	0.0	0.05	0.01	0.00	0.07	0.32	0.13	0.81	0.40
	0.2	0.00	0.00	0.03	0.06	0.59	0.37	0.20	1.64
	0.4	0.00	0.00	0.00	0.04	0.18	0.09	0.00	0.05
	0.6	0.00	0.00	0.00	0.00	0.14	0.02	0.00	0.00

Table 14 Improvement procedures - runtimes (in seconds)

var	*n*	API	3SW	INTER	INS
L	25	0.00	0.00	0.01	0.01
	50	0.01	0.02	0.07	0.04
	100	0.06	0.06	0.38	0.15
	250	0.24	0.24	8.88	2.02
	500	0.80	0.85	101.47	14.96
H	25	0.01	0.01	0.01	0.01
	50	0.01	0.02	0.09	0.05
	100	0.07	0.07	1.03	0.18
	250	0.28	0.30	36.29	2.53
	500	1.11	1.28	522.71	19.54

Table 15 Comparison with optimum objective function values

		n = 10		*n* = 15		*n* = 20	
var	heur	%dev	%opt	%dev	%opt	%dev	%opt
L	EQTP	3.15	16.25	7.14	7.50	4.18	5.63
	EQTP_Back	0.44	66.88	0.18	63.13	0.34	50.00
	DR_Back	0.22	75.00	0.11	70.00	0.17	55.00
	EQTP_Back_Ex	0.11	81.88	0.08	79.38	0.09	72.50
	DR_Back_Ex	0.04	94.38	0.04	90.00	0.06	83.13
	DR_Back_Ex + INS	0.01	95.63	0.01	93.13	0.04	87.50
	MA_IN_New	0.00	100.00	0.03	93.13	0.03	93.75
H	EQTP	28.97	3.13	25.36	0.63	33.33	0.00
	EQTP_Back	15.33	27.50	10.50	19.38	4.04	11.25
	DR_Back	7.67	41.25	3.49	24.38	2.26	13.13
	EQTP_Back_Ex	2.35	58.13	1.52	40.63	1.13	27.50
	DR_Back_Ex	1.24	77.50	0.81	60.63	0.89	44.38
	DR_Back_Ex + INS	0.31	84.38	0.24	73.13	0.41	53.75
	MA_IN_New	0.13	93.13	0.34	80.63	0.23	77.50

Table 16 Relative deviation from the optimum for instances with 20 jobs

		low var				high var			
heur	*T*	*R*=0.2	*R*=0.4	*R*=0.6	*R*=0.8	*R*=0.2	*R*=0.4	*R*=0.6	*R*=0.8
EQTP	0.0	5.15	6.24	4.07	5.05	7.02	21.56	45.36	51.47
	0.2	13.40	18.33	5.92	6.92	34.16	17.90	32.24	37.83
	0.4	0.07	0.10	0.26	1.35	3.19	5.73	11.09	55.88
	0.6	0.05	0.02	0.01	0.01	5.67	1.76	0.72	2.30
EQTP_Back	0.0	0.20	0.08	0.41	0.09	0.59	0.93	1.62	6.07
	0.2	2.24	0.07	0.11	0.31	11.22	2.34	1.19	6.25
	0.4	0.20	0.40	0.62	0.36	4.65	3.50	8.77	4.48
	0.6	0.19	0.09	0.07	0.04	1.68	3.18	2.83	5.28
DR_Back	0.0	0.20	0.08	0.41	0.09	0.59	0.93	1.62	5.85
	0.2	0.13	0.07	0.11	0.31	7.09	1.27	1.19	6.25
	0.4	0.14	0.13	0.50	0.36	1.51	1.30	2.38	1.54
	0.6	0.10	0.05	0.04	0.03	0.99	1.61	0.72	1.30
EQTP_Back_Ex	0.0	0.07	0.04	0.22	0.04	0.31	0.89	1.39	2.24
	0.2	0.11	0.05	0.11	0.31	2.29	0.60	0.70	3.31
	0.4	0.05	0.09	0.20	0.00	0.50	0.21	2.07	1.99
	0.6	0.01	0.01	0.02	0.03	0.62	0.39	0.09	0.42
DR_Back_Ex	0.0	0.07	0.04	0.22	0.04	0.31	0.89	1.39	2.24
	0.2	0.04	0.05	0.11	0.31	3.53	0.23	0.70	3.31
	0.4	0.01	0.01	0.00	0.00	0.05	0.10	0.09	0.29
	0.6	0.00	0.00	0.00	0.00	0.56	0.30	0.22	0.06
DR_Back_Ex + INS	0.0	0.07	0.04	0.10	0.00	0.20	0.61	1.06	0.91
	0.2	0.00	0.00	0.06	0.04	0.25	0.11	0.32	2.01
	0.4	0.01	0.01	0.00	0.00	0.05	0.10	0.09	0.27
	0.6	0.00	0.00	0.00	0.00	0.10	0.23	0.22	0.06
MA_IN_New	0.0	0.00	0.52	1.05	0.34	0.00	0.43	1.24	0.40
	0.2	0.00	0.01	0.01	0.10	0.01	0.00	0.17	1.29
	0.4	0.00	0.00	0.00	0.00	0.00	0.05	0.00	0.00
	0.6	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00

Table 17 Relative improvement over the EQTP heuristic

var	*n*	EQTP_Back	DR_Back	EQTP_Back_Ex	DR_Back_Ex	MA_IN	MA_IN_New
L	15	2.01	2.07	2.11	2.11	2.11	2.11
	25	1.18	1.22	1.25	1.25	1.25	1.26
	50	1.36	1.37	1.39	1.39	1.38	1.39
	100	1.27	1.28	1.28	1.28	1.27	1.28
	250	1.34	1.34	1.34	1.34	---	---
	500	1.49	1.49	1.49	1.49	---	---
	1000	1.58	1.58	1.58	1.58	---	---
H	15	3.83	6.58	7.77	7.82	7.96	7.97
	25	4.19	5.11	6.52	6.58	6.66	6.71
	50	4.51	4.72	5.13	5.12	5.15	5.23
	100	5.46	5.37	5.63	5.64	5.59	5.70
	250	5.87	5.90	5.92	5.94	---	---
	500	5.85	5.89	5.87	5.92	---	---
	1000	5.95	6.01	5.96	6.02	---	---

Table 18 Comparison with optimum objective function values

		n = 10		*n* = 15		*n* = 20	
var	heur	%dev	%opt	%dev	%opt	%dev	%opt
L	EQTP	1.31913	47.92	3.21971	35.42	1.14255	27.08
	EQTP_Back	0.30295	60.42	0.10193	55.00	0.19108	46.25
	DR_Back	0.13920	70.42	0.04843	64.17	0.07136	52.08
	EQTP_Back_Ex	0.00286	95.42	0.00568	95.00	0.00531	90.83
	DR_Back_Ex	0.00122	96.67	0.00489	96.67	0.00524	91.67
	DR_Back_Ex + INS	0.00018	99.17	0.00000	100.00	0.00089	96.25
	MA_IN	0.00023	99.92	0.00317	97.92	0.00675	96.08
	MA_IN_New	0.00000	100.00	0.00104	99.46	0.00131	99.08
H	EQTP	19.89999	20.00	13.25761	12.50	16.76017	7.92
	EQTP_Back	6.56151	29.17	4.76446	20.00	2.27440	15.83
	DR_Back	1.55379	42.08	1.56191	25.83	1.02715	17.08
	EQTP_Back_Ex	0.36876	80.83	0.25915	64.17	0.28980	57.50
	DR_Back_Ex	0.12821	86.67	0.19299	72.50	0.28267	64.17
	DR_Back_Ex + INS	0.03482	93.75	0.04141	86.25	0.07781	79.17
	MA_IN	0.00012	99.96	0.01979	96.21	0.07883	89.92
	MA_IN_New	0.00012	99.96	0.00765	97.29	0.00895	95.92

Table 19 Example 3 data

job (j)	1	2	3	4	5	6
p_j	34	76	9	65	30	89
d_j	209	309	352	398	422	243

Table 20 CT_Check and Adjust_T_IIT example

Idle time generated by procedure LB_IIT

i:	1	2	3	4	5	6
idle time:	120	0	0	0	0	0

Procedure CT_Check

i:	1	2	3	4	5	6
CT_C_i:	209	244	310	354	400	425

Procedure Adjust_T_IIT

					previous *idle*[*j*]						new *idle*[*j*]							chosen
i	previous *t*	CT_C_i	*P*	*reduct*	1	2	3	4	5	6	1	2	3	4	5	6	new *t*	job
6	423	425	303	0	120	0	0	0	0	0	120	0	0	0	0	0	423	5
5	393	400	273	0	120	0	0	0	0	0	120	0	0	0	0	0	393	4
4	328	354	208	0	120	0	0	0	0	0	120	0	0	0	0	0	328	3
3	319	310	199	9	120	0	0	0	0	0	111	0	0	9	0	0	310	2
2	234	244	123	0	111	0	0	9	0	0	111	0	0	9	0	0	234	6
1	145	209	34	0	111	0	0	9	0	0	111	0	0	9	0	0	145	1

(a) The DR_Back heuristic is used to select the chosen job at each iteration