Available online at www.sciencedirect.com

ScienceDirect

Computers & Industrial Engineering 52 (2007) 434–447

computers & industrial engineering

www.elsevier.com/locate/dsw

Dispatching heuristics for the single machine early/tardy scheduling problem with job-independent penalties ☆

Jorge M.S. Valente *

LIACC/NIAAD – Faculdade de Economia, Universidade do Porto, Rua Dr. Roberto Frias, 4200-464 Porto, Portugal

Received 7 October 2006; received in revised form 22 January 2007; accepted 1 February 2007
Available online 9 February 2007

Abstract

In this paper, we consider the single machine earliness/tardiness scheduling problem with job-independent penalties, and no machine idle time. Several dispatching heuristics are proposed, and their performance is analysed on a wide range of instances. The heuristics include simple scheduling rules, as well as a procedure that takes advantage of the strengths of each of those rules. We also consider early/tardy dispatching procedures, and a heuristic method based on existing adjacent precedence conditions. An improvement procedure that can be used to improve the schedules generated by the heuristics is also proposed.

The computational tests show that the best results are given by the early/tardy dispatching rules. These heuristics are also quite fast, and are capable of quickly solving even very large instances. The use of the improvement procedure is recommended, since it improves the solution quality, with little additional computational effort.
© 2007 Elsevier Ltd. All rights reserved.

Keywords: Scheduling; Single machine; Early/tardy; Job-independent penalties; Dispatching rules

1. Introduction

In this paper, we consider a single machine earliness/tardiness scheduling problem with job-independent penalties, and no machine idle time. Formally, the problem can be stated as follows. A set of n independent jobs $\{J_1, J_2, \ldots, J_n\}$ 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, $j = 1, 2, \ldots, n$, requires a processing time p_j and should ideally be completed on its due date d_j. Also, let h and w denote the job-independent earliness and tardiness penalties, respectively. For a given schedule, the earliness and tardiness of J_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 J_j. The objective is then to find a schedule that minimizes the sum of weighted earliness and weighted tardiness costs $\sum_{j=1}^{n}(hE_j + wT_j)$, subject to the constraint that no machine idle time is allowed.

☆ This manuscript was processed by Area Editor Maged M. Dessouky.
* Tel.: +351 22 557 11 00; fax: +351 22 550 50 50.
E-mail address: jvalente@fep.up.pt

0360-8352/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cie.2007.02.001

Scheduling models with a single machine may seem unrealistic. However, this scheduling environment does indeed occur in several activities (for a recent example, see Wagner, Davis, & Kher, 2002). Moreover, the performance of many production systems is quite often dictated by the quality of the schedules for a single bottleneck machine. Also, the analysis of single machine problems provides useful insights for scheduling more complex systems.

Scheduling models with both earliness and tardiness costs are compatible with the philosophy of just-in-time (JIT) production, which emphasizes producing goods only when they are needed. In a JIT setting, early jobs must be held in inventory until their due dates, while jobs that finish late may cause a customer, or a further stage in the production line, to delay or shutdown operations. Scheduling models with both early and tardy penalties are then compatible with the JIT philosophy, since jobs are scheduled to finish as close as possible to their due dates.

In this paper, we consider job-independent earliness and tardiness penalties. Job-independent earliness penalties are appropriate in production settings where the jobs are identical, or quite similar. In this situation, the holding costs will indeed be the same for all jobs. A job-independent tardiness penalty is also adequate when the jobs are similar, or when the company assigns the same importance to all customers.

We assume that machine idle time is not allowed. This assumption is actually appropriate for many production settings (for some specific examples, see Korman, 1994 and Landis, 1993). Indeed, when the capacity of the machine is limited when compared with the demand, the machine must be kept running in order to satisfy the customers' orders. Furthermore, this assumption is also justified when starting a new production run involves high setup costs or times, or when the machines have high operating costs.

The single machine early/tardy problem with job-independent penalties and no idle time was previously considered by Szwarc (1993) and Azizoglu, Kondakci, and Kirca (1991). Szwarc developed time-dependent precedence relations that hold for adjacent jobs in an optimal schedule. A branching scheme based on those adjacent ordering conditions was also proposed. Azizoglu et al. (1991) actually considered the equivalent problem $\sum_{j=1}^{n}((1-q)E_j + qT_j)$, with $0 < q < 1$. They proposed a lower bounding procedure, as well as a branch-and-bound algorithm.

The single machine earliness/tardiness problem with job-dependent penalties $\sum_{j=1}^{n}(h_jE_j + w_jT_j)$ and no idle time has also been considered by several authors, and both exact and heuristic approaches have been proposed. Among the exact approaches, lower bounds and branch-and-bound algorithms were presented by Abdul-Razaq and Potts (1988); Li (1997); Liaw (1999) and Valente and Alves (2005c). Among the heuristics, several dispatching rules and beam search algorithms were presented by Ow and Morton (1989) and Valente and Alves (2005a, 2005b). A neighbourhood search algorithm was also presented by Li (1997). A large number of papers consider scheduling models with earliness and tardiness penalties. Baker and Scudder (1990) provide an excellent survey of early/tardy scheduling problems, while Kanet & Sridharan (2000) give a review of scheduling models with inserted idle time that complements our focus on a problem with no machine idle time.

In this paper, we propose and analyse several dispatching heuristics. We consider three simple but widely used scheduling rules, as well as a heuristic that tries to take advantage of the strengths of each of those rules. We also present modified versions of early/tardy dispatching heuristics originally proposed for the problem with job-dependent earliness and tardiness penalties. A heuristic procedure that uses the dominance conditions proposed by Szwarc (1993) is also considered. We also propose an improvement procedure that incorporates Szwarc's adjacent ordering conditions.

The remainder of this paper is organized as follows. The heuristics are described in Section 2. In Section 3, we present the improvement procedure that was used to improve the schedule obtained by the heuristics. The computational results are given in Section 4, and some concluding remarks are provided in Section 5.

2. The heuristics

2.1. Simple dispatching rules

We consider three simple scheduling rules, namely the longest processing time (LPT), shortest processing time (SPT) and earliest due date (EDD) heuristics. The EDD rule schedules the jobs in non-decreasing order of their due dates, while the LPT (SPT) heuristic sequences the jobs in non-increasing (non-decreasing) order of their processing times. The time complexity of these rules is $O(n \log n)$, since they only require sorting.

These heuristics are considered for two major reasons. First, these rules are quite well-known and widely used in many production settings. Also, these heuristics were previously used by Azizoglu et al. (1991) in the upper bounding procedure of their branch-and-bound algorithm. Therefore, it seems sensible to include them for comparison purposes.

Second, these rules have some interesting properties for the early/tardy problem. The LPT (SPT) heuristic is adequate for problems where most jobs will be completed before (after) their due dates. Indeed, the LPT (SPT) sequence is optimal if it does not contain any tardy (early) jobs. The EDD rule, on the other hand, usually outperforms the LPT and SPT heuristics when the number of early and tardy jobs is relatively balanced. Therefore, each one of these rules can perform quite well, under the appropriate circumstances.

2.2. The LES heuristic

In this section, we present a heuristic (denoted by LES) that tries to take advantage of the strengths of the LPT, EDD and SPT rules. At each iteration, the LES heuristic uses one of these rules to choose the next job (in fact, LES stands for <u>L</u>PT–<u>E</u>DD–<u>S</u>PT). The LES heuristic selects, at each iteration, the rule that is expected to provide the best performance, under the characteristics of the current set of unscheduled jobs.

The LES heuristic classifies the current workload as early, critical or tardy. Let the slack of job j be defined as $s_j = d_j - t - p_j$, where t is the current time. When most jobs have large slacks, the current workload can be described as *early*. Conversely, a *tardy* load consists mainly of jobs with a negative slack. Finally, the load is said to be *critical* when several jobs have relatively low (positive) slacks, and are therefore at risk of becoming tardy. At each iteration, the LES heuristic analyses the current job load, and classifies that workload as either early, critical or tardy. Then, the LES procedure selects the LPT (EDD/SPT) rule when the load is early (critical/tardy).

We considered two versions of the LES heuristic. These versions differ only in the criterion used to classify the workload. In both versions, we first calculate a critical interval of slack values [0, max_slack]. The upper limit max_slack is calculated as $\max\_\text{slack} = \text{slack\_prop} * n_U * \overline{p}$, where n_U is the number of unscheduled jobs, $\overline{p}$ is the average processing time of those unscheduled jobs, and $0 < \text{slack\_prop} < 1$ is a user-defined parameter.

The LES_AS version calculates the average slack $\overline{s}$ of the remaining unscheduled jobs. The workload is then classified as early (critical/tardy) if $\overline{s} > \max\_\text{slack}(\overline{s} \in [0, \max\_\text{slack}]/\overline{s} < 0)$. In the LES_LP version, on the other hand, we first classify each unscheduled job as early, critical or tardy. A job j is said to be early (critical/tardy) if $s_j > \text{max\_slack}$ ($s_j \in [0, \text{max\_slack}]/s_j < 0$). The proportion of early, critical and tardy jobs is then calculated. Finally, the current workload is classified as early (critical/tardy) if the percentage of early (critical/tardy) jobs is the largest. The time complexity of both versions of the LES heuristic is $O(n^2)$.

2.3. Early/tardy dispatching rules

Ow and Morton (1989) developed two early/tardy dispatching rules, denoted as LINET and EXPET, for the problem with job-dependent earliness and tardiness penalties. In this section, we present modified versions of these rules, suitably adapted to the problem with job-independent penalties.

The LINET and EXPET dispatching rules calculate a priority index for each remaining job every time the machine becomes available, and the job with the highest priority is then selected to be processed next. Let $I_j(t)$ denote the priority index of job J_j at time t. The LINET heuristic then uses the following priority index $I_j(t)$:

$$I_j(t) = \begin{cases} \phi(1/p_j) & \text{if} \quad s_j \leqslant 0 \\ \phi(1/p_j) - (1/p_j)s_j/k\overline{p} & \text{if} \quad 0 < s_j < k\overline{p} \\ -(1-\phi)(1/p_j) & \text{otherwise,} \end{cases}$$

where $\phi = w/(h + w)$, k is a lookahead parameter and s_j and $\overline{p}$ are as previously defined.

The EXPET rule instead uses the following priority index:

$$I_j(t) = \begin{cases} \phi(1/p_j) & \text{if} \quad s_j \leqslant 0 \\ \phi(1/p_j)\exp\left[-\frac{1}{1-\phi}(s_j/k\overline{p})\right] & \text{if} \quad 0 < s_j < \phi k\overline{p} \\ [(1-\phi)(1/p_j)]^{-2}[\phi(1/p_j)-(1/p_j)s_j/k\overline{p}]^3 & \text{if} \quad \phi k\overline{p} \leqslant s_j < k\overline{p} \\ -(1-\phi)(1/p_j) & \text{otherwise,} \end{cases}$$

where ϕ, s_j, $\overline{p}$ and k are as previously defined.

The LINET and EXPET dispatching rules use a priority function that starts at $-(1-\phi)(1/p_j)$ when jobs are in no danger of being tardy ($s_j \geqslant k\overline{p}$), and then gradually increases to a maximum of $\phi(1/p_j)$ when jobs are on time or late ($s_j \leqslant 0$). Therefore, these rules reflect a priority that focuses on the tardiness cost of a job as its slack becomes small, while the earliness cost dominates when that slack is large.

The effectiveness of the LINET and EXPET heuristics depends on the value of the lookahead parameter k. This parameter is related to the number of competing critical jobs, i.e., the number of jobs that may clash each time a sequencing decision is to be made (for details, see Ow & Morton, 1989). We considered two different approaches for setting the value of k. In the first approach, the lookahead parameter is set at a fixed value. The heuristic versions that use a fixed value will be denoted simply as LINET and EXPET.

In the second approach, the value of k is calculated dynamically at each iteration. These versions will be denoted by appending "_vk" (standing for variable k) to the corresponding heuristic identifier. Each time a scheduling decision has to be made, the characteristics of the current job load are used to calculate an adequate value for the lookahead parameter. At each iteration, and just as previously mentioned in the discussion of the LES heuristics, each unscheduled job can be classified as early, critical or tardy. When most jobs are quite early or tardy, the problem is relatively easy. Therefore, the early and tardy jobs can then be jointly described as *non-critical*. When the current workload contains only or mostly non-critical jobs, the lookahead parameter can be appropriately set at a low value, denoted as k_L, since very few jobs will clash at a sequencing decision. When the workload consists mostly of critical jobs, however, the problem is much harder, because several jobs will soon become late. The number of competing critical jobs is then higher, and a higher value of k, denoted as k_H, is therefore adequate.

The value of k is then calculated as follows in the _vk versions. Just as previously described for the LES procedures, we first calculate a critical interval of slack values [0, max_slack]. Each job J_j is classified as critical if $s_j \in [0, \text{max\_slack}]$, and non-critical otherwise. The proportion of critical jobs *prop_crit* is then calculated. Finally, the lookahead parameter k is set at $k = \text{prop\_crit} \times k_H + (1 - \text{prop\_crit}) \times k_L$. The time complexity of both versions of the LINET and EXPET dispatching rules is $O(n^2)$.

2.4. Greedy heuristic

Szwarc (1993) developed time-dependent precedence relations for adjacent jobs in an optimal schedule. In order to apply these dominance conditions, the jobs are first renumbered in SPT order. Then, a critical time value t_{ij} is calculated for all pairs of jobs (i,j), with $i < j$. This critical value is such that job i precedes job j when the processing of this pair of jobs starts not earlier than t_{ij} (i.e., $t \geqslant t_{ij}$), while job j precedes job i when $t < t_{ij}$.

In this section, we present a heuristic (denoted by Greedy) that uses the dominance conditions proposed by Szwarc to calculate a priority value for each job. We considered two different versions of the Greedy heuristic. These versions differ only slightly in the calculation of the job priorities. Let L be a list that contains the SPT indexes of the jobs that have not yet been scheduled. Also, let $P(j)$ denote the priority of job j. The steps of the Greedy_v1 version are:

```
Step 1. Initialization:
        Renumber the jobs in SPT order;
        Calculate the critical values $t_{ij}$ for all pairs of jobs $(i,j)$, with $i < j$;
        Set $t = 0$ and $L = \{1, 2, \ldots, n\}$.
Step 2. Calculate the job priorities:
        Set $P(j) = 0$, for all $j \in L$;
```

```
    For all pairs of jobs $(i, j) \in L$, with $i < j$, do:
      If $t \geqslant t_{ij}$, set $P(i) = P(i) + 1$;
      Else, set $P(j) = P(j) + 1$.
Step 3. Select the next job:
    Schedule job $l$ for which $P(l) = \max\{P_j; j \in L\}$;
    Set $t = t + p_l$ and $L = L \backslash \{l\}$.
Step 4. Stopping condition:
    If $|L| = 1$, stop;
    Else, go to step 2.
```

In the Greedy_v2 version, Step 2 is instead given by:

```
Step 2. Calculate the job priorities:
    Set $P(j) = 0$, for all $j \in L$;
    For all pairs of jobs $(i, j) \in L$, with $i < j$, do:
      If $t \geqslant t_{ij}$
        set $P(i) = P(i) + |t - t_{ij}|$;
        set $P(j) = P(j) - |t - t_{ij}|$.
      Else
        set $P(i) = P(i) - |t - t_{ij}|$;
        set $P(j) = P(j) + |t - t_{ij}|$.
```

If $t \geqslant t_{ij}$, it seems better to schedule job i in the next position rather than job j. In fact, job i precedes job j in an optimum solution if they are scheduled in the next two positions in the sequence. Conversely, it seems preferable to schedule job j next when $t < t_{ij}$. In the Greedy_v1 version, the priority $P(j)$ of job j is simply the number of times job j is the preferred job for the next position when it is compared with all the other unscheduled jobs. In the Greedy_v2 version, for all pairs of jobs (i,j), with $i < j$, the priority of the preferred job is instead increased by $|t - t_{ij}|$, while the priority of the other job is decreased by that same value. The time complexity of both versions of the Greedy heuristic is $O(n^3)$.

3. The improvement procedure

In this section, we propose an improvement procedure that uses the time-dependent dominance conditions developed by Szwarc (1993) and described in the previous section. This procedure is applied as an improvement step, once the heuristics have generated a schedule. Let $[k]$ denote the index of the job in the kth position in the sequence. The improvement procedure can then be described as follows.

```
Step 1. Initialization:
    Renumber the jobs in SPT order;
    Calculate the critical values $t_{ij}$ for all pairs of jobs $(i, j)$, with $i < j$;
    Set $k = 1$ and $t = 0$.
Step 2. Apply the time-dependent dominance conditions:
    While $k < n$
      If $[k + 1]$ precedes $[k]$ at time $t$
        swap $[k]$ and $[k + 1]$;
        If $k > 1$
          Set $k = k - 1$ and $t = t - p_{[k]}$.
      Else
        Set $t = t + p_{[k]}$ and $k = k + 1$.
```

The improvement algorithm uses the time-dependent dominance conditions to perform any required adjacent pairwise interchanges. Therefore, the sequence generated by the procedure is locally optimal, in the sense that it cannot be further improved through adjacent swaps.

4. Computational results

In this section, we first present the set of problems used in the computational tests, and then describe the preliminary computational experiments. These experiments were performed to determine appropriate values for the parameters required by the LES, LINET and EXPET heuristics. Moreover, the performance of the alternative versions of the LES, LINET, EXPET and Greedy heuristics was also analysed in these initial experiments, in order to select the best-performing of those versions. Finally, we present the computational results. We first compare the heuristic procedures, and then analyse the effectiveness of the improvement algorithm. Throughout this section, and in order to avoid excessively large tables, we will sometimes present results only for some representative cases.

4.1. Experimental design

The computational tests were performed on a set of problems with 15, 20, 25, 30, 40, 50, 75, 100, 250, 500, 750, 1000, 1500 and 2000 jobs. These problems were randomly generated as follows. For each job J_j, an integer processing time p_j was generated from one of the two uniform distributions $[1, 10]$ and $[1, 100]$, to create low (L) and high (H) variability, respectively. An integer earliness penalty h and an integer tardiness penalty w were also generated from one of these two uniform distributions. The tardiness penalty weight $\phi = w/(h + w)$ was set at 0.1, 0.3, 0.5, 0.7 and 0.9. The earliness and tardiness penalties are then randomly generated in such a way that the desired ratio ϕ is obtained. For each job J_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, set at 0.0, 0.2, 0.4, 0.6, 0.8 and 1.0, and R is the range of due dates, set at 0.2, 0.4, 0.6 and 0.8.

For each combination of problem size n, processing time and penalty variability (var), ϕ, T and R, 50 instances were randomly generated. Therefore, a total of 6000 instances were generated for each combination of problem size and variability. All the algorithms were coded in Visual C++ 6.0, and executed on a Pentium IV – 2.8 GHz personal computer. Due to the large computational times that would be required, the Greedy heuristic was only applied to instances with up to 750 jobs.

4.2. Parameter adjustment tests

In this section, we describe the preliminary computational experiments. These initial experiments were performed to determine appropriate values for the parameters required by the LES, LINET and EXPET dispatching rules. The performance of the alternative versions of the LES, LINET, EXPET and Greedy heuristics was also analysed, in order to select the best-performing versions. A separate problem set was used to conduct these preliminary experiments. This test set included instances with 25, 50, 100, 250, 500, 1000 and 2000 jobs, and contained 5 instances for each combination of instance size, variability, ϕ, T and R. The instances in this smaller test set were randomly generated just as previously described for the full problem set.

We performed extensive computational tests to determine appropriate values for the parameters used by the LES, LINET, LINET_vk, EXPET and EXPET_vk heuristics. The LES_AS and LES_LP dispatching rules require a value for the slack_prop parameter. We considered the values $\{0.20, 0.25, \ldots, 0.80\}$, and computed the objective function value for each slack_prop value and each instance. An analysis of these results showed that the best values for slack_prop were in the range $[0.50, 0.70]$. We then decided to set slack_prop at 0.50 and 0.55, for the LES_AS and LES_LP versions, respectively, since these values consistently provided good results.

The LINET and EXPET heuristics require a value for the lookahead parameter k. We considered the values $\{0.5, 0.6, \ldots, 9.0\}$, and the objective function value was then computed for each value of k and each instance. The lookahead parameter k was then set at 4.0 in both the LINET and EXPET rules, since this value provided an adequate performance across all instances types.

The LINET_vk and EXPET_vk dispatching heuristics require a value for the parameters k_L, k_H and slack_prop. We considered the following values for these parameters:

$k_L : \{0.5, 1.0, 1.5, 2.0\}$,
$k_H : \{4.0, 4.5, \ldots, 8.0\}$,
slack_prop : $\{0.20, 0.25, 0.30, \ldots, 0.80\}$.

The LINET_vk and EXPET_vk rules were then applied to the test set instances for each combination of these parameter values. A thorough analysis of these results was conducted in order to determine values that provided an adequate performance for all instance types. The following values were then selected for both the LINET_vk and EXPET_vk heuristics: $k_L = 0.5$, $k_H = 5.0$ and slack_prop = 0.70.

We also analysed the performance of the alternative versions of the LES, LINET, EXPET and Greedy heuristics in these preliminary computational experiments, in order to select the best-performing versions. Therefore, we compared the following four ($h1$ vs $h2$) pairs of alternative heuristic versions: (LES_AS vs LES_LP), (LINET_vk vs LINET), (EXPET_vk vs EXPET) and (Greedy_v1 vs Greedy_v2).

In Table 1, we present the average of the relative improvements in objective function value provided by the $h1$ heuristic over its $h2$ counterpart (% imp), as well as the percentage number of times version $h1$ performs better (<), equal (=) or worse (>) than version $h2$. The relative improvement given by version $h1$ is calculated as $(h2\_ofv - h1\_ofv)/h2\_ofv \times 100$, where $h2$_ofv and $h1$_ofv are the objective function values of the appropriate heuristic versions.

The LES_AS version performs better than its LES_LP counterpart. The LES_AS heuristic gives an average objective function value that is about 1–2% better than the LES_LP version, and it provides better results for a somewhat larger number of instances. The dynamic LINET_vk and EXPET_vk heuristics provide a marginally superior performance when compared with their fixed lookahead parameter counterparts. In fact, the

Table 1
Heuristic version comparison

	n	Low var				High var			
		% imp	<	=	>	% imp	<	=	>
LES_AS vs LES_LP	25	1.09	36.17	43.67	20.17	1.00	38.17	38.17	23.67
	50	1.27	40.83	33.50	25.67	1.47	41.83	33.00	25.17
	100	1.87	42.67	29.17	28.17	1.81	45.83	23.83	30.33
	250	1.19	43.33	26.67	30.00	1.61	50.00	20.50	29.50
	500	1.13	44.67	23.83	31.50	1.37	50.33	15.33	34.33
	1000	0.58	45.33	20.83	33.83	1.72	52.33	13.33	34.33
	2000	0.02	42.67	22.33	35.00	1.75	52.33	12.17	35.50
LINET_vk vs LINET	25	1.11	68.17	16.17	15.67	1.10	71.33	11.33	17.33
	50	0.97	81.83	4.33	13.83	0.81	80.33	5.17	14.50
	100	0.64	84.50	2.67	12.83	0.54	81.83	1.33	16.83
	250	0.21	87.83	0.17	12.00	0.23	83.00	0.33	16.67
	500	0.11	88.67	0.33	11.00	0.12	84.50	0.67	14.83
	1000	0.06	87.00	0.50	12.50	0.06	81.00	2.17	16.83
	2000	0.02	85.00	1.17	13.83	0.02	80.17	2.67	17.17
EXPET_vk vs EXPET	25	1.13	64.00	12.67	23.33	1.23	68.00	10.17	21.83
	50	0.88	74.50	4.50	21.00	0.84	76.00	4.00	20.00
	100	0.63	79.67	1.33	19.00	0.62	80.33	1.33	18.33
	250	0.32	81.00	1.17	17.83	0.29	80.00	0.00	20.00
	500	0.18	81.83	0.33	17.83	0.12	80.50	0.33	19.17
	1000	0.07	79.67	1.00	19.33	0.05	80.50	1.00	18.50
	2000	0.04	77.83	1.33	20.83	0.03	78.50	1.67	19.83
Greedy_v1 vs Greedy_v2	25	12.97	94.00	0.00	6.00	13.33	95.00	0.00	5.00
	50	15.65	96.33	0.00	3.67	17.80	98.83	0.00	1.17
	100	17.19	96.17	0.00	3.83	20.43	100.00	0.00	0.00
	250	18.29	95.00	0.00	5.00	22.26	100.00	0.00	0.00
	500	18.72	95.50	0.00	4.50	22.99	100.00	0.00	0.00

relative improvement given by the dynamic versions is usually less than 1%. However, the LINET_vk and EXPET_vk heuristics provide better results for over 80% of the test instances.

The Greedy_v1 version clearly outperforms its Greedy_v2 alternative. Indeed, the Greedy_v1 heuristic provides a relative improvement in the objective function value of about 15–20%. Also, the Greedy_v1 version gives better results for over 95% (and in some cases, actually all) of the test instances. In the following sections, we will only present results for the better performing LES_AS, LINET_vk, EXPET_vk and Greedy_v1 versions.

4.3. Heuristic results

In this section, we present the computational results for the heuristic procedures. In Table 2, we give the average objective function value (ofv) for each heuristic, as well as the percentage number of times a heuristic provides the best result when compared with the other heuristics (% best). The average objective function values are calculated relative to the LINET_vk heuristic, and are therefore presented as index numbers. More precisely, these values are calculated as heur_ofv/linet_vk_ofv* 100, where heur_ofv and linet_vk_ofv are the average objective function values of the appropriate heuristic and the LINET_vk dispatching rule, respectively.

The best results are given by the LINET_vk, EXPET_vk and Greedy_v1 dispatching rules. In fact, these heuristics not only provide the lowest average objective function value, but also obtained the best results for a larger percentage of the instances. The LINET_vk and EXPET_vk give slightly better objective function values, while the Greedy_v1 heuristic achieves the best result for a somewhat higher number of instances. The LINET_vk heuristic is marginally superior to the EXPET_vk dispatching rule. This is a somewhat surprising result, since the EXPET heuristic provided better results in the computational tests performed by Ow and Morton (1989) for the problem with job-dependent penalties.

The LES_AS heuristic is outperformed by the early/tardy dispatching rules and the greedy procedure. Indeed, the LES_AS heuristic provides an average objective function value that is about 10% worse than the results given by the LINET_vk dispatching rule. The simple LPT, EDD and SPT rules perform rather poorly, giving results that are substantially worse than those of the early/tardy and greedy heuristics. Consequently, ignoring the earliness and/or tardiness penalties is quite costly in terms of solution quality. The LES_AS heuristic performs considerably better than either of the LPT, EDD and SPT rules. Therefore, it is indeed possible to obtain a significant performance improvement by selectively using these three simple heuristics.

Table 2
Heuristic results

Var	Heur	$n = 25$		$n = 100$		$n = 500$		$n = 2000$	
		Ofv	% best	Ofv	% best	Ofv	% best	Ofv	% best
L	EDD	140.15	0.92	142.80	0.97	143.32	0.43	143.49	0.45
	LPT	177.28	0.77	182.14	0.00	183.73	0.00	184.27	0.00
	SPT	172.86	1.52	178.17	0.07	179.66	0.68	179.41	0.83
	LES_AS	109.69	6.28	110.40	8.95	110.32	11.42	110.35	11.03
	LINET_vk	100.00	40.03	100.00	33.87	100.00	26.85	100.00	49.75
	EXPET_vk	100.50	30.33	100.01	26.08	99.99	24.92	100.00	43.47
	Greedy_v1	100.87	60.80	100.83	42.20	100.21	39.48	–	–
H	EDD	143.80	1.57	147.08	1.50	147.68	1.43	147.95	1.42
	LPT	184.08	0.13	190.53	0.00	192.21	0.00	192.91	0.00
	SPT	183.09	0.48	189.93	0.10	191.97	0.80	191.09	0.83
	LES_AS	110.55	4.95	111.26	10.02	111.24	13.63	111.10	14.32
	LINET_vk	100.00	40.62	100.00	33.77	100.00	27.18	100.00	46.23
	EXPET_vk	100.53	26.30	100.05	24.42	100.00	27.20	100.00	41.23
	Greedy_v1	101.20	53.47	100.87	35.55	100.21	32.38	–	–

Table 3
Heuristic runtimes (in seconds)

Var	Heur	$n = 500$	$n = 750$	$n = 1000$	$n = 1500$	$n = 2000$
L	EDD	0.0001	0.0001	0.0001	0.0002	0.0004
	LPT	0.0001	0.0002	0.0002	0.0004	0.0005
	SPT	0.0001	0.0002	0.0002	0.0004	0.0006
	LES_AS	0.0011	0.0026	0.0043	0.0096	0.0173
	LINET_vk	0.0069	0.0153	0.0267	0.0607	0.1101
	EXPET_vk	0.0073	0.0177	0.0301	0.0658	0.1160
	Greedy_v1	1.0805	5.0165	–	–	–
H	EDD	0.0001	0.0001	0.0002	0.0002	0.0003
	LPT	0.0001	0.0002	0.0002	0.0004	0.0005
	SPT	0.0001	0.0002	0.0002	0.0004	0.0005
	LES_AS	0.0011	0.0023	0.0039	0.0089	0.0155
	LINET_vk	0.0066	0.0150	0.0267	0.0596	0.1083
	EXPET_vk	0.0073	0.0166	0.0291	0.0642	0.1138
	Greedy_v1	1.1032	5.0815	–	–	–

In Table 3, we present the heuristic runtimes (in seconds). The Greedy_v1 heuristic is computationally demanding, and can only be used for small and medium size instances. The other heuristic procedures are quite fast, even for the largest instances. The simple LPT, EDD and SPT rules are the most efficient, since they only require $O(n \log n)$ time. The LES_AS heuristic and the early/tardy dispatching rules are more computationally demanding, given their $O(n^2)$ time complexity. Nevertheless, these heuristics are also extremely fast, since they are capable of solving even quite large instances with 2000 jobs in around 0.1 s on a personal computer. The LINET_vk and EXPET_vk are then the heuristic procedures of choice, since they do not only provide the best results, but are also computationally quite efficient.

4.4. Improvement step results

In this section, we present the computational results for the improvement procedure. In Table 4, we give the average of the relative improvements in the objective function value, calculated as (ofv – imp_ofv)/ofv*100, where ofv and imp_ofv are the objective function values of the appropriate heuristic before and after the application of the improvement step, respectively.

We also performed a test to determine if the difference in distribution between the objective function values before and after the improvement step is statistically significant. Given that exactly the same problems were

Table 4
Improvement procedure results

Var	Heur	$n = 25$	$n = 50$	$n = 100$	$n = 250$	$n = 500$	$n = 1000$	$n = 2000$
L	EDD	27.59	28.29	28.21	27.94	27.73	27.60	27.54
	LPT	43.34	44.29	44.95	45.48	45.75	45.88	46.04
	SPT	26.51	24.05	21.78	19.69	19.01	18.69	18.55
	LES_AS	12.08	12.25	12.24	11.62	11.34	11.09	10.93
	LINET_vk	2.20	1.19	0.62	0.26	0.14	0.07	0.04
	EXPET_vk	2.79	1.52	0.77	0.31	0.16	0.08	0.04
	Greedy_v1	0.39	0.28	0.15	0.07	0.04	–	–
H	EDD	29.37	30.05	30.06	29.66	29.34	28.96	28.64
	LPT	45.07	46.18	46.98	47.52	47.76	47.95	48.06
	SPT	26.61	24.00	21.72	19.14	17.78	16.69	15.93
	LES_AS	12.58	12.97	12.60	11.89	11.28	10.82	10.39
	LINET_vk	2.26	1.20	0.63	0.27	0.14	0.07	0.04
	EXPET_vk	2.87	1.53	0.79	0.32	0.16	0.08	0.04
	Greedy_v1	0.49	0.38	0.22	0.10	0.05	–	–

used to obtain these objective function values, a paired-samples test is appropriate. Since not all the hypothesis of the paired-samples *t*-test were met, the non-parametric Wilcoxon test was selected. The significance values of this test were always equal to 0.000. The significance value gives the probability of obtaining results as extreme as the one observed when the equal distribution hypothesis is true. Therefore, low significance values mean that the differences between the objective function values are statistically significant, while high significance values support the equal distribution hypothesis.

From Table 4, we can see that the improvement step is effective in reducing the objective function value. The Wilcoxon test significance values also indicate that the differences in distribution between the heuristic results before and after the improvement step are statistically significant. The relative improvement decreases with the quality of the schedules generated by the heuristic procedures. In fact, the relative improvement is usually quite marginal for the best-performing heuristics, although the LINET_vk and EXPET_vk rules still benefit from a 1–2% improvement for the small size instances. The results provided by the LES_AS procedure, on the other hand, are improved by over 10%. Finally, the worst-performing LPT, EDD and SPT heuristics benefit the most from the improvement step. These simple procedures were quite far in solution quality from the other heuristics, and therefore had a much larger room for improvement.

In Table 5, we present the effect of the ϕ, T and R parameters on the relative objective function value improvement for the EDD heuristic and instances with 100 jobs. Tables 6–8 provide the same results for

Table 5
Relative improvement for the EDD heuristic and instances with 100 jobs

ϕ	T	Low var				High var			
		$R=0.2$	$R=0.4$	$R=0.6$	$R=0.8$	$R=0.2$	$R=0.4$	$R=0.6$	$R=0.8$
0.1	0.0	27.41	23.33	20.78	18.01	29.14	24.58	20.55	18.34
	0.2	29.01	24.92	20.99	19.21	29.12	26.00	22.88	19.77
	0.4	27.21	26.18	23.66	23.06	30.45	27.58	24.72	24.23
	0.6	30.96	30.23	32.58	36.59	32.75	33.29	34.41	38.72
	0.8	33.24	36.77	35.31	34.01	35.68	39.26	38.76	35.93
	1.0	29.41	28.59	27.13	26.11	32.87	30.90	30.33	29.32
0.3	0.0	27.48	23.68	20.75	16.93	29.05	24.40	20.41	18.57
	0.2	28.12	24.80	21.12	18.22	29.47	24.90	22.41	19.69
	0.4	28.92	26.50	23.78	21.50	31.08	27.63	25.11	22.93
	0.6	32.20	33.16	36.15	37.78	34.03	35.17	36.74	39.97
	0.8	33.38	37.06	36.25	33.81	37.17	39.22	38.19	36.45
	1.0	28.94	28.84	28.11	26.04	31.62	31.07	30.29	28.56
0.5	0.0	26.84	23.92	19.12	17.44	27.93	23.46	20.58	17.45
	0.2	27.12	24.05	21.27	18.51	29.27	25.34	22.09	19.07
	0.4	29.00	27.11	24.32	21.92	30.36	28.93	25.87	22.56
	0.6	31.77	34.38	36.59	39.49	34.18	36.74	39.63	40.62
	0.8	32.75	36.52	36.24	34.18	35.56	39.25	37.93	35.83
	1.0	28.97	28.65	27.69	26.24	31.58	30.54	30.25	28.98
0.7	0.0	27.05	23.42	20.08	17.33	28.81	24.68	20.53	16.97
	0.2	28.61	23.58	20.98	17.80	29.41	24.69	21.37	19.29
	0.4	31.10	29.61	24.96	21.62	32.96	30.20	26.56	22.31
	0.6	32.46	36.36	37.21	40.18	35.05	38.20	40.65	42.77
	0.8	33.71	36.95	35.94	33.91	37.41	40.22	39.20	35.97
	1.0	29.25	29.07	27.91	27.14	32.20	30.89	30.36	29.02
0.9	0.0	26.26	22.22	19.80	17.21	28.40	24.18	20.03	17.05
	0.2	27.48	23.74	19.67	17.63	29.19	25.43	21.87	17.76
	0.4	31.35	32.59	28.75	21.63	35.96	33.66	30.49	22.04
	0.6	32.81	36.50	38.26	38.97	35.94	38.72	40.81	43.14
	0.8	33.96	36.68	35.63	33.94	36.52	40.86	38.91	36.84
	1.0	28.84	28.70	27.88	26.66	31.73	30.67	30.55	29.25

Table 6
Relative improvement for the LPT heuristic and instances with 100 jobs

ϕ	T	Low var				High var			
		$R = 0.2$	$R = 0.4$	$R = 0.6$	$R = 0.8$	$R = 0.2$	$R = 0.4$	$R = 0.6$	$R = 0.8$
0.1	0.0	0.46	1.45	2.76	4.42	0.63	1.85	3.24	5.25
	0.2	5.42	12.45	15.36	18.78	5.63	12.94	17.86	21.36
	0.4	12.78	25.52	40.41	54.67	13.77	24.92	40.47	56.01
	0.6	30.59	47.33	73.84	90.47	31.86	48.40	69.25	88.09
	0.8	52.87	64.20	66.90	68.82	53.08	68.01	70.45	71.46
	1.0	45.44	45.08	44.87	45.52	49.79	50.03	49.27	50.73
0.3	0.0	0.70	2.41	4.66	6.71	1.08	3.02	5.20	8.02
	0.2	11.67	20.05	24.58	28.30	11.66	20.65	25.77	29.28
	0.4	27.46	41.07	54.25	66.13	29.27	41.22	54.72	66.78
	0.6	45.21	57.04	72.92	82.50	46.12	57.69	70.45	84.70
	0.8	53.68	59.99	60.39	60.56	55.49	62.56	64.70	63.95
	1.0	44.86	44.83	44.94	43.82	48.61	47.80	48.08	47.93
0.5	0.0	1.12	3.55	5.91	10.30	1.58	4.55	8.43	11.89
	0.2	17.64	28.34	33.64	37.69	19.89	31.28	37.27	41.25
	0.4	36.66	49.37	65.02	76.12	40.26	51.65	65.31	75.29
	0.6	49.38	60.29	73.18	81.13	52.06	62.23	71.97	83.22
	0.8	53.10	58.71	59.35	59.09	55.13	62.18	62.33	62.63
	1.0	45.33	45.29	44.27	42.92	48.55	48.06	47.53	46.68
0.7	0.0	2.35	7.21	13.98	19.32	3.38	9.79	15.93	21.24
	0.2	32.99	46.31	53.22	56.86	33.53	46.77	53.51	58.68
	0.4	48.16	61.47	74.53	84.53	51.10	61.49	73.43	83.78
	0.6	52.63	62.14	72.50	79.66	55.28	63.81	72.50	80.43
	0.8	53.57	58.41	59.10	57.62	55.89	62.28	62.35	60.67
	1.0	45.57	45.23	44.17	44.69	48.79	48.22	47.32	47.03
0.9	0.0	7.58	19.50	31.24	42.55	11.80	27.00	37.97	50.79
	0.2	51.88	70.15	76.22	79.81	55.66	74.35	79.95	82.76
	0.4	54.61	67.93	80.98	91.49	58.73	69.59	81.06	90.81
	0.6	54.63	63.83	73.47	80.10	57.67	66.38	73.27	80.43
	0.8	53.42	58.18	57.93	57.36	56.05	62.03	62.11	60.25
	1.0	45.09	45.00	44.67	43.61	48.27	48.10	47.87	47.12

the LPT, SPT and LINET_vk heuristics, respectively. The effect on the relative improvement is significantly different for the several heuristics. For the EDD rule, the relative improvement in the objective function value is of a somewhat similar magnitude for all instance types. When the LPT rule is considered, however, the relative improvement is much larger for the higher values of the tardiness factor T. This is to be expected, since the LPT rule performs better when most jobs are early. When the tardiness factor is larger, most jobs will instead be tardy. Therefore, the performance of the LPT heuristic deteriorates, and the potential for improvement consequently increases.

On the contrary, the relative improvement is decreasing with the tardiness factor T for the SPT rule. Indeed, this heuristic performs better when most jobs are tardy. As the tardiness factor decreases, the proportion of early jobs increases, so there is more room for improvement. For the remaining heuristics, the relative improvement if higher when T is equal to 0.4 and 0.6, and then decreases as the tardiness factor approaches its extreme values. Indeed, the improvement step actually provides little improvement when T is equal to 0.0 or 1.0. This result is also to be expected, since these heuristics consider both earliness and tardiness costs. These procedures are then more likely to be closer to the optimum for the extreme values of T. In fact, for a tardiness factor of 0.0 (1.0) most jobs will be early (late), and the early/tardy problem is then easier. For the intermediate values of T, the problem becomes much harder, since there is a greater balance between the number of early and tardy jobs.

Table 7
Relative improvement for the SPT heuristic and instances with 100 jobs

ϕ	T	Low var				High var			
		$R = 0.2$	$R = 0.4$	$R = 0.6$	$R = 0.8$	$R = 0.2$	$R = 0.4$	$R = 0.6$	$R = 0.8$
0.1	0.0	42.64	37.73	33.79	29.02	45.15	39.01	33.60	29.11
	0.2	43.29	37.42	31.51	27.94	43.41	37.82	32.54	26.76
	0.4	40.41	34.15	28.68	26.40	41.28	34.12	27.52	23.89
	0.6	36.73	28.95	26.46	28.50	36.85	27.19	22.47	22.99
	0.8	25.45	28.06	30.80	33.36	23.70	20.60	25.27	25.76
	1.0	5.42	12.53	19.78	23.88	6.31	12.54	17.03	21.68
0.3	0.0	42.59	36.98	33.11	27.62	44.99	39.00	33.02	29.14
	0.2	41.57	36.96	31.88	25.91	42.55	36.33	31.42	26.36
	0.4	36.64	31.25	26.66	24.53	39.32	31.65	25.75	22.81
	0.6	29.13	24.58	20.44	20.56	29.92	22.85	17.87	18.20
	0.8	11.83	14.20	16.58	19.77	13.30	11.78	14.33	16.86
	1.0	1.36	3.96	6.54	8.00	1.69	4.03	6.22	8.74
0.5	0.0	42.53	37.49	32.29	29.28	43.86	38.58	33.65	28.64
	0.2	40.73	36.37	31.64	27.09	41.70	36.25	30.87	26.30
	0.4	33.97	29.55	25.98	22.48	35.96	29.84	24.67	22.34
	0.6	22.70	18.66	16.82	18.90	23.10	18.59	15.51	16.41
	0.8	7.19	8.35	11.77	12.42	7.31	7.45	9.26	11.12
	1.0	0.75	1.98	3.54	4.84	0.91	2.03	3.56	4.70
0.7	0.0	42.53	37.22	32.49	27.74	44.04	38.05	32.38	28.19
	0.2	39.44	34.22	30.74	25.52	40.43	35.27	30.51	26.14
	0.4	28.53	24.15	21.79	19.13	30.83	25.69	22.19	20.31
	0.6	13.61	11.60	12.20	14.56	14.81	13.29	12.39	12.06
	0.8	3.10	4.79	6.58	7.82	3.97	4.85	6.13	7.80
	1.0	0.41	1.15	2.04	2.82	0.49	1.19	2.13	3.08
0.9	0.0	41.41	36.47	32.83	28.61	43.18	38.31	33.86	29.04
	0.2	33.64	31.72	27.20	26.15	34.34	33.05	31.31	27.04
	0.4	16.01	15.30	15.33	16.17	16.96	17.77	17.87	20.42
	0.6	5.40	6.13	7.21	9.59	5.91	7.21	8.54	10.51
	0.8	1.31	2.63	3.91	4.78	1.56	2.66	4.14	5.09
	1.0	0.27	0.77	1.31	1.78	0.33	0.73	1.23	1.81

In Table 9, we give the computation time (in seconds) required by the improvement step. The additional computational effort is not very significant, since the improvement step is executed in less than 0.5 s for even the quite large instances with 2000 jobs. Therefore, the use of the improvement step is recommended, since it is effective in reducing the objective function value.

The improvement procedure does not alter the heuristic recommendation provided in the previous section. In fact, the LINET_vk and EXPET_vk heuristics still provide the best results after the application of the improvement step. However, the difference in performance between the several heuristics is greatly reduced by the improvement procedure, since the relative improvement is much higher for the worst performing heuristics. The LINET_vk or EXPET_vk dispatching rules, followed by the application of the improvement step, are then the heuristic procedure of choice.

5. Conclusion

In this paper, we considered the single machine scheduling problem with job-independent earliness and tardiness penalties and no machine idle time. We analysed the performance of several dispatching rules on a wide range of instances, and provided recommendations on which heuristic to use. The heuristics included simple scheduling rules, as well as a procedure that takes advantage of the strengths of each of those rules. We also

Table 8
Relative improvement for the LINET_vk heuristic and instances with 100 jobs

ϕ	T	Low var				High var			
		$R = 0.2$	$R = 0.4$	$R = 0.6$	$R = 0.8$	$R = 0.2$	$R = 0.4$	$R = 0.6$	$R = 0.8$
0.1	0.0	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.01
	0.2	0.03	0.06	0.04	0.03	0.03	0.04	0.03	0.02
	0.4	0.04	0.10	0.19	0.27	0.05	0.10	0.15	0.18
	0.6	0.09	0.22	0.77	1.26	0.12	0.26	0.58	1.36
	0.8	0.18	0.19	0.19	0.16	0.25	0.40	0.32	0.26
	1.0	0.00	0.01	0.01	0.02	0.03	0.04	0.04	0.04
0.3	0.0	0.00	0.01	0.01	0.01	0.00	0.01	0.01	0.01
	0.2	0.17	0.18	0.09	0.07	0.18	0.14	0.08	0.05
	0.4	0.29	0.61	0.89	0.97	0.34	0.58	0.82	0.75
	0.6	0.50	0.92	2.15	2.68	0.52	0.94	1.83	3.12
	0.8	0.57	0.65	0.59	0.53	0.63	0.90	0.79	0.63
	1.0	0.02	0.05	0.07	0.07	0.06	0.08	0.10	0.11
0.5	0.0	0.00	0.01	0.01	0.02	0.01	0.01	0.02	0.02
	0.2	0.35	0.35	0.16	0.10	0.39	0.35	0.18	0.13
	0.4	0.53	1.11	1.88	1.99	0.65	1.19	1.76	1.76
	0.6	0.61	1.30	2.54	3.32	0.77	1.43	2.38	4.36
	0.8	0.52	0.83	0.64	0.61	0.68	0.96	0.84	0.80
	1.0	0.03	0.06	0.08	0.09	0.07	0.09	0.12	0.14
0.7	0.0	0.01	0.02	0.02	0.04	0.01	0.03	0.04	0.06
	0.2	0.71	0.72	0.33	0.26	0.67	0.67	0.35	0.26
	0.4	0.78	1.60	2.74	3.66	0.77	1.53	2.64	3.23
	0.6	0.62	1.24	2.18	2.97	0.62	1.23	2.13	3.13
	0.8	0.45	0.63	0.67	0.56	0.49	0.78	0.75	0.65
	1.0	0.04	0.07	0.09	0.12	0.06	0.10	0.12	0.12
0.9	0.0	0.02	0.07	0.10	0.11	0.03	0.13	0.14	0.19
	0.2	1.10	1.24	0.54	0.47	0.92	1.37	0.61	0.55
	0.4	0.63	1.51	3.10	4.76	0.58	1.31	2.49	4.11
	0.6	0.44	0.88	1.48	2.40	0.36	0.77	1.30	1.93
	0.8	0.26	0.46	0.45	0.45	0.26	0.47	0.46	0.45
	1.0	0.03	0.06	0.07	0.08	0.04	0.07	0.09	0.10

Table 9
Improvement procedure runtimes (in seconds)

Var	Heur	$n = 250$	$n = 500$	$n = 1000$	$n = 1500$	$n = 2000$
L	EDD	0.0041	0.1081	0.0958	0.1692	0.2805
	LPT	0.0043	0.0440	0.0901	0.1788	0.3151
	SPT	0.0045	0.0424	0.0916	0.2014	0.3505
	LES_AS	0.0035	0.0568	0.0915	0.1396	0.2303
	LINET_vk	0.0034	0.0272	0.0703	0.1287	0.2187
	EXPET_vk	0.0036	0.0400	0.0655	0.1284	0.2118
	Greedy_v1	0.0041	0.0339	–	–	–
H	EDD	0.0063	0.0391	0.0950	0.1790	0.2984
	LPT	0.0062	0.0438	0.0992	0.1936	0.3539
	SPT	0.0058	0.0474	0.1025	0.2150	0.3681
	LES_AS	0.0065	0.0560	0.1118	0.1506	0.2429
	LINET_vk	0.0038	0.0279	0.0712	0.1357	0.2269
	EXPET_vk	0.0084	0.0249	0.0823	0.1351	0.2236
	Greedy_v1	0.0064	0.0348	–	–	–

considered early/tardy dispatching rules, and a heuristic method based on the precedence relations proposed by Szwarc (1993). Extensive computational experiments were performed to determine adequate values for the parameters required by some of the heuristics. An improvement algorithm that uses the adjacent ordering conditions developed by Szwarc was also proposed.

The best results were given by the LINET_vk, EXPET_vk and Greedy_v1 dispatching rules. The Greedy_v1 heuristic, however, is computationally demanding, and can only be used for small and medium size instances. The other heuristic procedures were quite fast, and are capable of quickly solving even very large instances. The use of the improvement step is recommended, since it can improve the solution quality with little additional computational effort. The LINET_vk or EXPET_vk dispatching rules, followed by the application of the improvement step, are then the heuristic procedure of choice.

Acknowledgement

The author would like to thank an anonymous referee for several comments that were used to improve this paper.

References

Abdul-Razaq, T., & Potts, C. N. (1988). Dynamic programming state-space relaxation for single machine scheduling. *Journal of the Operational Research Society, 39*, 141–152.

Azizoglu, M., Kondakci, S., & Kirca, O. (1991). Bicriteria scheduling problem involving total tardiness and total earliness penalties. *International Journal of Production Economics, 23*, 17–24.

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

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

Korman, K. (1994). A pressing matter. Video, (pp. 46–50).

Landis, K. (1993). Group technology and cellular manufacturing in the Westvaco Los Angeles VH department. Project report in IOM 581, School of Business, University of Southern California.

Li, G. (1997). Single machine earliness and tardiness scheduling. *European Journal of Operational Research, 96*, 546–558.

Liaw, C.-F. (1999). A branch-and-bound algorithm for the single machine earliness and tardiness scheduling problem. *Computers & Operations Research, 26*, 679–693.

Ow, P. S., & Morton, T. E. (1989). The single machine early/tardy problem. *Management Science, 35*, 177–191.

Szwarc, W. (1993). Adjacent orderings in single-machine scheduling with earliness and tardiness penalties. *Naval Research Logistics, 40*, 229–243.

Valente, J. M. S., & Alves, R. A. F. S. (2005a). Filtered and recovering beam search algorithms for the early/tardy scheduling problem with no idle time. *Computers & Industrial Engineering, 48*, 363–375.

Valente, J. M. S., & Alves, R. A. F. S. (2005b). Improved heuristics for the early/tardy scheduling problem with no idle time. *Computers & Operations Research, 32*, 557–569.

Valente, J. M. S., & Alves, R. A. F. S. (2005c). Improved lower bounds for the early/tardy scheduling problem with no idle time. *Journal of the Operational Research Society, 56*, 604–612.

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