Expert Systems With Applications 119 (2019) 376–386

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Heuristics for scheduling jobs in a permutation flow shop to minimize total earliness and tardiness with unforced idle time allowed ☆,☆☆

Jeffrey Schaller [a,*], Jorge M.S. Valente [b]

[a] Department of Business Administration, Eastern Connecticut State University, 83 Windham St., Willimantic, CT 06226 -2295
[b] LIAAD – INESC TEC, Faculdade de Economia, Universidade do Porto, Rua Dr. Roberto Frias, 4200-464 Porto, Portugal

ARTICLE INFO

Article history:
Received 25 July 2018
Revised 19 October 2018
Accepted 5 November 2018
Available online 7 November 2018

Keywords:
Scheduling
Heuristics
Flow Shop
Earliness and Tardiness

ABSTRACT

This paper considers the problem of scheduling jobs in a permutation flow shop with the objective of minimizing total earliness and tardiness. Unforced idle time is considered in order to reduce the earliness of jobs. It is shown how unforced idle time can be inserted on the final machine. Several dispatching heuristics that have been used for the problem without unforced idle time were modified and tested. Several procedures were also developed that conduct a second pass to develop a sequence using dispatching rules. These procedures were also tested and were found to result in better solutions.

1. Introduction

Organizations place importance on timely delivery of products while at the same time minimizing inventory. These elements have become an important indicator of an operation's effectiveness. Both early delivery and tardy delivery of products are indicators of poor supply chain execution. If products are delivered early unnecessary inventory results and this requires space, cash and resources by the customer. Products delivered after their due date can result in lost sales and the loss of customer good will. Total earliness and tardiness is a measure of performance for the quality of a schedule. To address the above considerations, in this paper, we have an objective that sums the penalties for earliness and tardiness for a set of jobs processed in a flow shop. When jobs maintain the same sequence on all of the machines in a flow shop it is called a permutation flow shop. Most research on flow shops use this assumption. There are two reasons for permutation flow shops. The computational effort to develop schedules is simplified and it would require extra physical handling to change the sequence of jobs in the middle of the shop. We only consider permutation schedules in this paper.

Formally, suppose there is a set of n jobs to be processed in a permutation flow shop with M machines. Let d_j be the due date of job j ($j = 1, \ldots, n$). Let p_{jm} and C_{jm} represent the processing time and completion time of job j ($j = 1, \ldots, n$) on machine m ($m = 1, \ldots, M$). The earliness of job j, E_j, is defined as: $E_j = max\ \{d_j - C_{jM}, 0\}$, for $j = 1,\ldots, n$ and the tardiness of job j, T_j, is defined as: $T_j = max\ \{C_{jM} - d_j, 0\}$, for $j = 1,\ldots, n$. The objective function, Z, can be expressed as: $Z = \sum_{j=1}^{n} E_j + T_j$.

Baker (1974) defines a performance measure as regular as follows. "A performance measure Z is regular if a) the scheduling objective is to minimize Z, and b) Z can increase only if at least one of the completion times in the schedule increases." Because of the inclusion of earliness in the objective of this problem, the decrease of a completion time could cause the objective to increase therefore this problem has a non-regular objective. Since the problem has a non-regular objective, unforced idle time can be inserted to reduce the earliness of some jobs thereby improving the objective. Forced idle time is required whenever a machine becomes available but the next job to be processed on that machine is not yet ready for processing. Additional idle time is unforced idle time. However, to the best of our knowledge, previous research has not considered unforced idle time for this problem. In this research, we consider schedules with unforced inserted idle time. We denote the job to be sequenced in position j as $[j]$. If $C_{[0]1} = 0$ then $C_{[j]1} = C_{[j-1]1} + I_{[j]1} + p_{[j]1}$ and $C_{[j]m} = max\ \{C_{[j]m-1}, C_{[j-1]m}\} + I_{[j]m} + p_{[j]m}$ for $m = 2, \ldots, M$, where $I_{[j]m}$ is the

☆ Declarations of interest: none
☆☆ This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

* Corresponding author.
E-mail addresses: schallerj@easternct.edu, schallerj@ecsu.ctstateu.edu (J. Schaller), jvalente@fep.up.pt (J.M.S. Valente).

https://doi.org/10.1016/j.eswa.2018.11.007
0957-4174/

unforced idle time inserted before the job in position *[j]* on machine *m*.

Many papers consider earliness and tardiness penalties. The first comprehensive survey covering the early papers for early/tardy scheduling was by Baker and Scudder (1990). Hoogeveen (2005) provides a survey of multicriteria scheduling which includes research that is more recent on problems with earliness and tardiness penalties. The single machine environment has the greatest amount of research on the early/tardy objective. Recent research was summarized by Valente (2009) for the single machine environment with no idle time allowed and an early/tardy objective. Kanet and Sridharan (2000) reviewed scheduling models when inserted idle time is allowed. Several of the papers consider the problem of how idle time can be optimally inserted if given a sequence for the single-machine problem. Fry, Armstrong, and Blackstone (1987) were the first to address this problem. They formulated the problem as a linear program. Special characteristics of the problem were used by Davis and Kanet (1993) and Yano and Kim (1991) to develop more efficient timetabling procedures. Branch-and-bound procedures were developed by Davis and Kanet (1993), Kim and Yano (1994), and Schaller for the single-machine problem for finding an optimal sequence and schedule. Dominance conditions were developed for the single-machine problem with inserted idle time by Kim and Yano (1994), Szwarc (1993), and Schaller (2007) in order to eliminate partial sequences within a branch-and-bound algorithm.

We know of nine papers that investigate flow shop environments with objectives that consider earliness and tardiness costs. The use of unforced idle time to reduce earliness costs was not considered in these papers. Zegordi, Itoh, and Enkawa (1997) and Rajendran (1999) were the earliest papers. Zegordi et al. (1997) considered the objective of minimizing the sum of weighted earliness and tardiness for a permutation flow shop; a simulated annealing algorithm was presented for the problem. Rajendran (1999) considered scheduling a kanban flow shop with the objective of minimizing the sum of weighted flowtime, weighted tardiness and weighted earliness of the kanban containers and presented heuristics for the problem. An optimal procedure was presented by Moslehi, Mirzaee, Vasei, and Azaron (2009) for a two-machine flow shop to minimize the sum of the maximum earliness and the maximum tardiness. The problem of scheduling a permutation flow shop when the jobs all have the same due date with earliness and tardiness penalties was addressed by Chandra, Mehta, and Tirupati (2009). Branch-and-bound algorithms were presented by Madhushini, Rajendran, and Deepa (2009) for a number of objectives one of which was the minimization earliness and tardiness. A genetic algorithm was proposed by Schaller and Valente (2013b) and the algorithm was compared with five other neighborhood search and metaheuristics for a permutation flow shop to minimize total earliness and tardiness. A variable neighborhood search heuristic was developed by M'Hallah (2014) and Fernandez-Viagas, Dios, and Framinan (2016) developed a constructive heuristic and local searches for the problem Schaller and Valente (2013b) investigated. Schaller and Valente (2013a) incorporated family setups into the permutation flow shop problem to minimize total earliness and tardiness. Several metaheuristics were compared for this problem and it was found a genetic algorithm worked best.

The next section explains how to insert unforced idle time to minimize total earliness and tardiness given a permutation sequence. In section three dispatching heuristics for the problem are described and in section four these heuristics are tested. Section five proposes sets of multiple sequences dispatching heuristics for the problem. Section six describes the computational tests and presents the results for the procedures described in section five and section seven concludes the paper.

2. Inserting unforced idle time to reduce earliness in order to minimize the total earliness and tardiness of a sequence

This paper only considers permutation schedules. Therefore to define a solution a job sequence is required. Since we are also considering the use of unforced inserted idle time to reduce the earliness of jobs we also need to determine if and where to insert unforced idle time for a given sequence of jobs to determine a schedule. The sequence and the schedule of unforced idle time define a solution.

For a given sequence of jobs in which the job to be sequenced in position j is denoted as *[j]*, the completion time of job *[j]* on machine M will determine the earliness or tardiness of the job in position j of the sequence. As shown in section 1 the completion time $C_{[j]M} = max\ \{C_{[j]M-1}, C_{[j-1]M}\} + I_{[j]M} + p_{[j]M}$, where $C_{[0]M} = 0$. Therefore a lower bound on the start time of the job in position j is $max\ \{C_{[j]M-1}, C_{[j-1]M}\}$ and a lower bound on the completion time of the job in position j is $max\ \{C_{[j]M-1}, C_{[j-1]M}\} + p_{[j]M}$ ($I_{[j]M} = 0$). A single-machine timetabling procedure for inserting idle time into a given sequence for minimizing total earliness and tardiness (Davis & Kanet, 1993; Fry et al., 1987; Kim & Yano, 1994) can be used with the constraint that the jobs cannot start before $max\ \{C_{[j]M-1}, C_{[j-1]M}\}$ for $[j] = 1,\ldots, n$. Inserting unforced idle time on machines 1 through $M-1$ need not be considered as doing so can only tighten the above constraint and possibly increase a solution's objective value.

3. Dispatching heuristics tested

Several heuristics were tested for the problem. The first set of heuristics that are described, are simple dispatching heuristics that are variants of dispatching heuristics that have been used in other environments but are modified to consider the flow shop environment with unforced idle time on the final machine as discussed in the previous section. These heuristics use simple indexes to select an unscheduled job that is appended to an initial partial sequence. The indexes consider trial completion times and the due dates of jobs in order to obtain sequences with low total earliness and tardiness. Once a complete sequence is obtained a timetabling algorithm is used to insert unforced idle time as described in the previous section in order to minimize the total earliness and tardiness of the obtained sequence.

In the heuristics that follow let S be the current partial schedule and $C_{jM}(S)$ be the completion time of job $j \notin S$ if j is scheduled at the end of S. Let $s_j(S)$ be the slack of job $j \notin S$ if j is scheduled at the end of S, where $s_j(S) = d_j - C_{jM}(S)$. Additionally, let $t_m(S)$ be the current availability time of machine m under schedule S. For convenience, the current time on the first machine will also be denoted by t, so $t = t_1(S)$. Finally, let $P_j(S) = C_{jM}(S) - t$ be the total time (total processing time plus any eventual forced idle time) between the start and finish of job $j \notin S$ if j is scheduled at the end of S.

In all the heuristics described in this section a sequence is developed by using a priority index to select an unscheduled job that is appended to an initial partial sequence. As each job is selected the trial completion times of the candidate jobs are calculated without considering unforced idle time. After a complete sequence is developed then a timetabling algorithm is used to insert unforced idle time and the solution's total earliness and tardiness is calculated.

We use an example with five jobs ($n = 5$) and three machines ($M = 3$) to demonstrate some of the calculations for the dispatching heuristics in this section. Table 1 has the data for this example, which includes the processing time for each job on each machine and each job's due date.

Table 1
Example problem with five jobs and three machines.

Machine	Job				
	1	2	3	4	5
1	8	6	4	17	19
2	20	8	13	12	14
3	19	15	2	11	19
Due Date	79	84	77	85	82

Table 2
Completion times for the EDD sequence with and without unforced idle time.

Machine	Job				
Without unforced idle time	3	1	5	2	4
1	4	12	31	37	54
2	17	37	51	59	71
3	19	56	75	90	101
With unforced idle time					
3	44	63	82	97	108

Table 3
Priority indexes using the MDD rule for the example problem.

Priority index (MDD)	Iteration				
Job	1	2	3	4	5
1	79	79			
2	84	84	84	90	101
3	77				
4	85	85	85	86	
5	82	82	82		
Selected job	3	1	5	4	2

3.1. Simple dispatching heuristics

Four simple dispatching heuristics are considered: earliest due date (EDD), modified due date (MDD), minimum slack rule (SLK), and minimum slack per work (SLK/P).

The earliest due date (EDD) rule was first proposed by Jackson (1955). This rule schedules the jobs in non-decreasing order of their due dates d_j. Using the data in Table 1 for the example problem the job sequence using this rule is 3-1-5-2-4. Table 2 shows the schedule for this sequence without using unforced idle time and when unforced idle time is added to the schedule on machine 3. After unforced idle is used the total earliness and tardiness of this sequence is 85.

In the modified due date (MDD) heuristic (Baker & Bertrand, 1982; Vepsalainen & Morton, 1987), at each iteration we select the job with the minimum value of the modified due date.

$$\mathrm{MDD}_j(S) = \max\{d_j, t + P_j(S)\} = \max\{d_j - -t, P_j(S)\}.$$

Using the data in Table 1 for the example problem the job sequence using this rule is 3-1-5-4-2. Table 3 shows the priority index (MDD_j (S)) during each iteration using this rule. After unforced idle is used the total earliness and tardiness of the sequence for this rule is 81.

The minimum slack (SLK) rule (Panwalkar & Iskander, 1977; Vepsalainen & Morton, 1987) chooses, at each iteration, the job with the minimum slack

$s_j\ (S) = d_j - C_{jM}\ (S)$. The minimum slack per required time (SLK/P) (Panwalkar & Iskander, 1977; Vepsalainen & Morton, 1987) selects, at each iteration, the job with the minimum value of the ratio $\mathrm{SLK}/P_j\ (S) = s_j\ (S)/P_j\ (S)$. Using the data in Table 1 for the example problem for the minimum slack (SLK) rule the job sequence is 5-1-2-4-3. Table 4 shows the priority index (SLK_j (S)) during each iteration using this rule. After unforced idle is used the total earliness and tardiness of the sequence for this rule is 75.

Table 4
Priority indexes using the SLK rule for the example problem.

Priority index (SLK)	Iteration				
Job	1	2	3	4	5
1	32	7			
2	55	17	-3		
3	58	23	3	-12	-23
4	45	22	2	-13	
5	30				
Selected job	5	1	2	4	3

Using the minimum slack per required time rule (SLK/P) for the example problem the sequence 5-1-2-3-4 is obtained and after using unforced idle time the total earliness and tardiness for this rule is 66. Table 5 shows the slack (s_j), time required (P_j) and the priority index during each iteration of this rule (s_j/P_j/index).

3.2. Dispatch rules with more advanced indexes

In this section we present dispatching rules that utilize priority indexes that consider a variety conditions that may be present.

3.2.1. LIN-ET Rules (LIN1 and LIN2)

The first two rules are based on the LIN–ET procedure proposed in (Ow & Morton, 1989) for the weighted single machine problem. These rules, which will be denoted by LIN1 and LIN2, choose, at each iteration, the job with the largest value of the following priority indexes (ties are broken by selecting the lowest numbered job):

$$\mathrm{LIN1}_j(S) = \begin{cases} \dfrac{1}{P_j(S)} & \text{if } s_j(S) \le 0 \\ \dfrac{1}{P_j(S)} - \dfrac{s_j(S)}{slk\_thr} * \dfrac{2}{P_j(S)} & \text{if } 0 < s_j(S) < \mathrm{slk\_thr} \\ -\dfrac{1}{P_j(S)} & \text{if } s_j(S) \ge \mathrm{slk\_thr} \end{cases}$$

and

$$\mathrm{LIN2}_j(S) = \begin{cases} \dfrac{1}{P_j(S)} & \text{if } s_j(S) \le 0 \\ \dfrac{1}{P_j(S)} - -s_j(S) * \left(\dfrac{1}{slk\_thr * P_j(S)} + \dfrac{1}{P_j(S)} \right) & \text{if } 0 < s_j(S) < \mathrm{slk\_thr} \\ -\dfrac{s_j(S)}{P_j(S)} & \text{if } s_j(S) \ge \mathrm{slk\_thr} \end{cases}$$

Where slk_thr, which stands for “slack threshold”, is a parameter meant to represent a value such that slacks which are greater or equal to that value are considered large.

In the first branch of the priority index, identical in both heuristics, a job is late or on time if scheduled next. When one or more such jobs exist, LIN1 and LIN2 select the job using a shortest time rule, in line with various heuristics for (earliness and) tardiness problems (Kenneth R. Baker, 1974; Ow & Morton, 1989; Smith, 1956).

In the third branch, the job has a large slack, and is quite early. If all jobs are quite early, LIN1 chooses the job using a longest time rule, again in line with several heuristics for early/tardy problems

Table 5
Priority indexes using the SLK/P rule for the example problem.

$s_{j/Pj}$/index	Iteration				
Job	1	2	3	4	5
1	32/47/0.681	7/53/0.132			
2	55/29/1.897	17/48/0.354	−3/60/−0.050		
3	58/19/3.053	23/35/0.657	3/47/0.064	−12/56/−0.214	
4	45/40/1.125	22/44/0.500	2/56/0.036	−13/65/−0.200	−15/63/−0.238
5	30/52/0.577				
Selected job	5	1	2	3	4

(Ow & Morton, 1989; Valente, 2007; Valente & Alves, 2005). LIN2, on the other hand, selects the job with the minimum slack per required time. Finally, the middle branch performs a linear interpolation between the priority values corresponding to $s_j(S) = 0$ and $s_j(S) =$ slk_thr. Such an interpolation was first performed in the LIN–ET procedure (Ow & Morton, 1989).

The slk_thr parameter is calculated as follows. At each iteration, the slack threshold is set equal to slk_thr $= w * (C^{LB}_{\max}(S) - t)$, where $C^{LB}_{\max}(S)$is a lower bound on the completion time of the last job on the final machine (makespan), given the current schedule S, and $0 \leq w \leq 1$ is a user-defined parameter. The lower bound is calculated using an adaptation of the procedure proposed in (Taillard, 1993). Indeed, the procedure of (Taillard, 1993) assumes that all machines are available at time zero. Since a lower bound is calculated at each iteration of LIN1 and LIN2, it was necessary to adapt the procedure to deal with non-zero machine availability times, which occur as jobs are scheduled.

The lower bound $C^{LB}_{\max}(S)$is calculated as follows. Let $\text{Bef_M}_i(S) = \min_{j \notin S}(t + \sum_{k=1}^{i-1} p_{kj})$be a lower bound on the time needed before reaching machine i, since it considers the availability time on the first machine, plus the minimum, over all unscheduled jobs, of the sum of the processing times on the machines that precede i. Also let $\text{TPT_M}_i(S) = \sum_{j \notin S} p_{ij}$be the total processing time required by all unscheduled jobs on machine i. Furthermore, let $\text{Aft_M}_i(S) = \min_{j \notin S}(t + \sum_{k=i+1}^{m} p_{kj})$be a lower bound on the time required after machine i, since it considers the minimum, over all unscheduled jobs, of the sum of the processing times on the machines that follow i.

For each machine i, a lower bound on the makespan is then calculated as $C^{LB}_{\max}(S) = \max\{Bef\_M_i(S), t_i\} + TPT\_M_i(S) + Aft\_M_i(S)$. In the original bound of (Taillard, 1993), all machine availability times were zero. Given a partial schedule, and/or machine availability times different from zero, two adaptations were required. The first was including the availability time of the first machine in $Bef\_M_i(S)$. The second was using the maximum between the bound on the time needed before reaching machine i and the availability time of this machine: $\max\{Bef\_M_i(S), t_i\}$. The lower bound on the makespan $C^{LB}_{\max}(S)$is equal to the maximum of all machine lower bounds: $C^{LB}_{\max}(S) = \max_i(C^{LB}_{\max\_M_i}(S))$.

The procedures formed using these indexes are referred to as LIN1 and LIN2 in this paper. Using the LIN1 rule for the example problem the sequence is 5-1-3-2-4 with a total earliness and tardiness of 56 and using the LIN2 rule the sequence is 5-1-2-3-4 with a total earliness and tardiness of 66. Tables 6 and 7 show the slack (s_j), time required (P_j) and the priority index during each iteration of these rules. In these tables we have also added a row that shows the estimated makespan and slk_thr during each iteration using a value for v of 0.8 for LIN1 and 0.2 for LIN2.

3.2.2. *Fernandez-Viagas et al. (2016) constructive heuristic*

A constructive heuristic for the problem without unforced idle time was developed by Fernandez-Viagas et al. (2016) and is described here. In this heuristic a sequence is built from the beginning to the end by picking a job to be sequenced first and then adding one job at a time to a partial sequence. An index is used to select the next job to be added to the partial sequence. The job with the lowest earliness, if sequenced first, is selected as the first job in the sequence. If there are ties for lowest earliness then the job with the lowest weighted idle time (defined below) is selected as the first job. For the remaining iterations of the procedure (k= 1 to n − 1) the problem is classified according to the due dates of the jobs that remain to be sequenced into one of three cases: 1) tight due dates, 2) loose due dates, and 3) due dates that are not tight or loose. Based on this classification the index used to select the next job changes.

To help define the classifications, if k jobs have been scheduled (kth iteration) let NT_k be the number of jobs from the unscheduled set $(n - k)$ that would be tardy if scheduled next. Let NE_k be the number of jobs from the unscheduled set that would have an earliness that is greater than $(n - k) * c$ if no unforced idle time were used, where c is a user defined parameter. The classifications are as follows.

1) Tight due dates – if the fraction of tardy jobs is greater than a, where a is a user defined parameter: $NT_k/(n - k) \geq a$.
2) Loose due dates – this case is divided into two subcases:

Subcase A) if there are at least four jobs still to be scheduled ($n - k > 3$), all the unscheduled jobs will be early if scheduled next, and $NE_k = n - k$.
Subcase B) there are at least four jobs still to be scheduled ($n - k > 3$), all the unscheduled jobs will be early if scheduled next, and $b * (n - k) \leq NE_k < n - k$, where b is a user defined parameter.

3) If the above criteria are not met then the due dates are not tight or loose.

Let EI_{jk} be the index for job j if scheduled in the last position of a partial sequence with k jobs. Each time a job is to be selected, the job with the lowest index among the unscheduled jobs is selected. Let E_{jk} equal the earliness of job j if scheduled in the last position of a partial sequence with k jobs and IT_{jk} be the weighted idle time of the candidate jobs: $IT_{jk} = \sum_{m=2}^{M} m * max\{C_{jm-1} - C_{jm}, 0\}/(m - 1 + (k*(M - m + 1)/(n - 2)))$. The indexes are:

1) Tight due dates. $EI_{jk} = (n - k - 2)/4 * IT_{jk} + C_{jM}$.
2) Loose due dates

Subcase A) Extremely loose due dates. $EI_{jk} = -(n - k - 2)/4 * IT_{jk} - C_{jM}$.
Subcase B) Moderately loose due dates. $EI_{jk} = -(n - k - 2)/4 * IT_{jk} - C_{jM} + E_{jk}$.

3) Due dates are neither tight nor loose. $EI_{jk} = E_{jk}$.

The procedure formed using this index is referred to as FV in this paper. Using the data in Table 1 for the example problem with the following parameter values, a = 0.90, b = 0.55 and c = 230,

Table 6
Priority indexes using the LIN1 rule for the example problem.

s_j/P_j/index	Iteration				
Job	1	2	3	4	5
1	32/47/0.000	7/53/0.015			
2	55/29/–0.025	17/48/0.010	–3/60/0.017	–5/58/0.017	
3	58/19/–0.043	23/35/0.008	3/47/0.019		
4	45/40/–0.010	22/44/0.007	2/56/0.017	–4/58/0.017	–15/63/0.016
5	30/52/0.001				
M_i/slk_thr	80/64.00	99/64.00	100/58.40	100/55.20	100/50.40
Selected job	5	1	3	2	4

Table 7
Priority indexes using the LIN2 rule for the example problem.

s_j/P_j/index	Iteration				
Job	1	2	3	4	5
1	32/47/-0.681	7/53/-0.121			
2	55/29/-1.897	17/48/-0.354	-3/60/0.017		
3	58/19/-3.053	23/35/-0.657	3/47-/0.047	- 12/56/0.018	
4	45/40/-1.125	22/44/-0.500	2/56/-0.020	-13/65/0.015	-15/63/0.016
5	30/52/-0.577				
M_i/slk_thr	80/16.00	99/16.00	100/14.60	100/13.40	100/12.60
Selected job	5	1	2	3	4

Table 8
Priority indexes using the FV procedure for the example problem.

I_{jk}/E_{jk}/index	Iteration				
Job	0	1	2	3	4
1	66/0.00/32/32.00	1.29/7/7.00			
2	39.00/55/55.00	0.00/17/17.00	0.00/0/0.00		
3	37.50/58/58.00	0.00/23/23.00	0.00/3/3.00	0.00/0/0.00	
4	94.50/45/45.00	5.40/22/22.00	0.00/2/2.00	0.00/0/0.00	0.00/0/0.00
5	106.50/30/30.00				
Case		3	3	3	3
Selected job	5	1	3	2	4

the job sequence using this rule is 5-1-2-3-4. Table 8 shows the Weighted Idle Time (IT_{jk}), Earliness (E_{jk}) and the index (EI_{jk}) for each job as well as the case used during each iteration using this rule. After unforced idle time is used the total earliness and tardiness of the sequence for this rule is 66.

4. Computational test of the dispatching heuristics

The proposed algorithms are tested on randomly generated instances of various levels of number of jobs and number of machines and under various conditions of due date range and tightness.

4.1. Data

The dispatching heuristic procedures described in section three were tested on instances of various levels of the number of jobs and number of machines for nine sets of distributions of due date range and tightness. Each instance set consists of 10 instances. The instances within a set have the same number of jobs and machines, and the due dates for the jobs are generated using the same distribution. Eight levels of number of jobs (n) to be scheduled were tested: $n = 15, 20, 25, 30, 40, 50, 75$ and 100. Three levels of number of machines (M) were tested: $M = 5, 10$ and 20. A uniform distribution over the integers 1 and 100 was used to generate the processing times of the jobs for each machine.

To randomly generate due dates for the jobs a uniform distribution over the integers $MS\ (1 - r - R/2)$ and $MS\ (1 - r + R/2)$ was used, where MS is an estimated makespan found for the problem using the makespan lower bound proposed in Taillard (1993), and R and r represent parameters referred to as due date range and tardiness factors. Three levels of due date range (R) were tested: $R = 0.2, 0.6$ and 1.0 and three levels of due date tightness (r) were tested: $r = 0.0, 0.2$ and 0.4. The levels of r are fairly low and represent low levels of due date tightness. The reason for these choices is that if due dates are tight most jobs would be tardy and unforced idle time would not be needed. These levels of R and r result in nine sets of due date parameters for each n and M combination.

A second set of 10 instances for each of the sets of parameters described above were created to use in preliminary tests to determine the parameter settings for the LIN1, LIN2, and FV procedures. The procedures were coded in Turbo Pascal and were tested on a Dell Inspiron 1525 GHz Lap Top computer.

4.2. Results for the dispatching heuristics

The measure of performance used to evaluate the dispatching procedures for the instances is percentage deviation ($\%\ Dev$) of the total earliness and tardiness of the solution generated by each procedure from the lowest total earliness and tardiness generated by the procedures. $\%\ Dev = [(Z_h - Z_B)/ Z_h] * 100$, where Z_B = the lowest total earliness and tardiness of the solutions generated by the seven procedures, and Z_h = the total earliness and tardiness of the solutions generated by the dispatching heuristic procedures (EDD, MDD, LIN1, LIN2, SLK, SLKP, FV).

Using the second set of instances the parameter w, used to calculate the slack threshold used in the LIN1 and LIN2 procedures,

Table 9
% deviation from best solution among dispatching heuristics for $M = 5$.

n	Procedure						
	EDD	MDD	LIN1	LIN2	SLK	SLKP	FV
15	20.22	6.79	12.44	22.83	33.64	38.03	15.68
20	20.65	10.98	9.59	18.19	28.83	30.98	11.95
25	21.91	8.27	8.10	13.88	25.28	27.39	11.39
30	19.35	7.17	7.34	14.35	24.10	26.50	10.36
40	16.97	6.82	12.44	13.12	19.06	21.71	9.11
50	14.89	7.34	17.63	13.83	19.11	21.97	7.02
75	12.11	4.66	20.01	9.23	13.83	15.41	8.10
100	12.94	5.82	44.53	14.65	13.30	21.11	8.64
Ave.	17.38	7.23	16.51	15.01	22.14	25.39	10.28

Table 10
% deviation from best solution among dispatching heuristics for $M = 10$.

n	Procedure						
	EDD	MDD	LIN1	LIN2	SLK	SLKP	FV
15	27.45	10.82	12.42	23.03	45.01	44.86	17.03
20	25.95	11.33	13.37	25.75	36.73	41.22	15.55
25	26.95	14.27	13.04	18.72	32.54	33.15	10.50
30	26.67	13.02	11.15	18.58	31.49	32.88	8.05
40	21.91	10.71	11.05	14.95	27.02	27.56	8.37
50	18.70	9.28	10.72	14.90	23.73	25.45	8.02
75	17.00	9.39	13.73	13.54	19.55	22.82	6.50
100	15.24	9.66	18.78	13.19	16.07	18.73	5.74
Ave.	22.48	11.05	13.03	17.83	29.02	30.83	9.97

Table 11
% deviation from best solution among dispatching heuristics for $M = 20$.

n	Procedure						
	EDD	MDD	LIN1	LIN2	SLK	SLKP	FV
15	35.10	13.35	14.75	22.60	47.09	47.56	12.28
20	30.28	14.55	17.80	23.24	41.09	39.72	9.87
25	29.56	11.06	13.25	24.08	41.83	42.66	12.48
30	29.53	14.69	14.94	21.35	36.38	36.08	8.44
40	24.65	14.29	12.85	17.51	28.58	29.16	5.46
50	25.83	15.28	14.56	18.53	27.87	29.23	3.81
75	20.66	13.62	13.80	17.45	25.41	25.37	5.15
100	18.96	12.56	12.50	15.21	20.89	22.87	4.52
Ave.	26.82	13.68	14.31	20.00	33.64	34.08	7.75

and the parameters a, b and c used in the FV procedure were determined. The w selected for LIN1 was 0.80, for LIN2 was 0.20. The values selected for the FV procedure were $a = 0.90$, $b = 0.55$, and $c = 230$. The values for the a and b parameters selected for the FV procedure were the same as those used by Fernandez-Viagas et al. (2016) when unforced idle time was not considered. The value selected for the parameter c was different however, and it was found that this parameter greatly affects the results.

Tables 9, 10 and 11 show the *% Dev* for each dispatching procedure for each level of number of jobs to be sequenced (n) as well as the averages across all the levels of jobs. Table 9 shows the results for $M = 5$, table 10, for $M = 10$ and table 11, for $M = 20$.

The results show that the MDD procedure was generally best when $m = 5$ but the FV procedure was generally best for $M = 10$ and 20. There were also a couple of combinations of n and M where LIN1 was best. The FV procedure was the only procedure to have a % DEV that was less than 11% for all the problem sizes. It was also best for 14 of the 24 combinations of n and M and was best when $n > 25$ and $M = 10$ or 20. It appears that as instance sizes become larger the relative performance of the FV procedure improves relative to the other procedures. The SLK and SLKP procedures were consistently the worst performing procedures with average *% Devs* over 20% for all three machine levels.

Table 12
% deviation from best solution by r for $n = 50$ and $M = 10$.

r	Procedure						
	EDD	MDD	LIN1	LIN2	SLK	SLKP	FV
0.0	5.44	4.21	5.85	18.15	12.05	19.59	11.56
0.2	16.51	4.92	3.56	11.16	21.60	21.58	11.55
0.4	34.16	18.73	22.74	15.39	37.55	35.18	0.95

Table 13
% deviation from best solution by R for $n = 50$ and $M = 10$.

R	Procedure						
	EDD	MDD	LIN1	LIN2	SLK	SLKP	FV
0.2	16.36	8.35	6.81	2.92	17.85	11.12	6.95
0.6	19.26	6.17	5.75	8.03	23.09	22.33	4.70
1.0	20.48	13.33	19.60	33.75	30.26	42.90	12.42

In order to show the effect of the due date range (R) and tardiness factor (r) on the results tables 12 and 13 are presented. Table 12 shows the *% Dev* by due date tardiness factor (r) for $n = 50$ and $M = 10$.

The results by due date tardiness factor (r) show that the MDD procedure was best for $r = 0.0$, LIN1 for $r = 0.2$, and FV for $r = 0.4$. The MDD, FV and LIN2 procedures were more consistent than the other procedures and these three procedures had a % Dev that was less than 20% for all three parameter values. This parameter does impact the results. The FV procedure was much better than the other procedures when $r = 0.4$ (the highest value) with a % Dev of less than 1 % whereas the MDD and LIN1 procedure were best when $r = 0.0$ and 0.2 with % Devs of less than 6%. When r is small the use of unforced idle becomes more important and explains why the results vary by the setting of this parameter.

Table 13 shows the *% Dev* by due date range factor (R) for $n = 50$ and $M = 10$.

The results show that the MDD and FV procedures were the most consistent across the due date range parameter (R) and each procedure had a % Dev that was less than 15 for all three levels. The LIN2 procedure was best when $R = 0.2$ but was not as good as the range of due dates increased. The SLK and SLKP procedures also seemed to deteriorate relative to the other procedures as R increases.

We checked to see if the differences in the mean % Dev for each pair of heuristics are significantly different by performing a repeated measures ANOVA test. There were interaction effects present for the number of jobs and machines. Therefore, the test was performed separately for each combination of M and n. The results showed that there is a significant effect regarding the heuristic used for all combinations of M and n (the hypothesis that all the heuristics perform similarly was rejected). Post-hoc tests were then conducted between each pair of heuristics, using the Bonferroni correction for the number of comparisons being performed. A 5% level of significance was used.

The improved performance of the MDD procedure compared to the EDD, SLK, and SLKP procedures was found to be significant for all combinations of n and M. The improved performance of the FV procedure compared to the EDD, SLK, and SLKP procedures was found to be significant in 20 of the 24 combinations of n and M, including all eight levels of n when $M = 20$. The statistical tests also confirm that as M gets larger the relative performance of the FV procedure improves. When $M = 10$ or 20, the improved performance of the FV procedure compared to the LIN2 procedure was statistically significant for all eight levels of n, and for $M = 20$, the results for the FV procedure was statistically different than those of the LIN1 and MDD procedures for half of the levels of n.

All of these procedures are very fast and were able to average less than 0.2 seconds per instance for all instance sizes. Therefore we do not present tables for the computation times for these procedures.

5. Multiple sequence heuristics

In this section heuristics that consider more than a single sequence are described. In the heuristics described in section three a sequence was developed by appending jobs to an initial partial sequence. When a job was to be selected the trial completion times of the candidate jobs and indexes were calculated without considering unforced idle time. After a complete sequence was developed then a timetabling algorithm was used to insert unforced idle time and then the solution's total earliness and tardiness was calculated. One of the problems with developing a sequence before inserting unforced idle time is that the current time used in the various indexes could be quite inaccurate. This is quite likely to be the case when due dates are relatively loose, since unforced idle time will then be inserted. The inaccuracy of the current time may then frequently cause the wrong job to be selected.

In order to overcome this problem, a two-stage procedure is used. In the first stage, we use a dispatching rule to generate an initial sequence, and then insert unforced idle time to optimize the total earliness and tardiness for this sequence.

In the second step, we start at the beginning of the initial sequence and select one job at a time, possibly developing multiple additional different sequences. More specifically, during an iteration of the second stage, a job is chosen for the current position in the sequence. When choosing this job, we use the current start time on the final machine, which includes any inserted unforced idle time.

Whenever a job is selected for a position, the remaining unscheduled jobs are kept in their order in the initial sequence, thus potentially creating a new sequence. Unforced idle time is inserted, if necessary, to optimize the total earliness and tardiness corresponding to this sequence.

Therefore, multiple sequences, with corresponding optimal unforced inserted idle times, can be evaluated during the procedure. The best of all these sequences is returned at the end of the two-stage procedure.

5.1. Initial sequence

We used three dispatching heuristics to generate the initial sequence: EDD, MDD, and FV. The EDD rule was chosen because it is simple and very efficient. The MDD and FV procedures were chosen because they generally performed best and also are very efficient. We then use the timetabling procedure to insert unforced idle time into the sequence and retain the objective value found as the initial incumbent value. The start time for processing on the last machine for the first job in the initial sequence becomes the initial current time for the second pass using other dispatching procedures.

5.2. Multiple sequence procedures

These procedures conduct a second pass adding one job at a time. When the job for position k is to be selected we would have $k - 1$ jobs that were selected using one of the indexes described below. These jobs would be in the first $k - 1$ positions of the current sequence. We also have $n - k - 1$ jobs remaining that are in these positions of the current sequence in the order of the initial sequence. We also have an estimate of the unforced idle time before each position on the final machine and an estimated start time. When we are selecting the job for position k the current time is the start time for processing on the last machine for the job in the first position of the initial sequence portion of the sequence.

We then obtain a trial completion time for each candidate job by taking the maximum of two completion time estimates. The first estimate is equal to the current time on the final machine plus the job's processing time on the final machine. This estimate takes inserted unforced idle time into account, but disregards the previous machines.

The second estimate is equal to the completion time of the job on the final machine, if no unforced idle time is used. This second estimate takes all the machines into account, but disregards unforced idle time.

The first estimate is more likely to be accurate when due dates are loose, while the second is likely to be better when due dates are tight (in this case, little or no unforced idle time is used). The trial completion time then takes both estimates into account, thereby adjusting itself to the characteristics of the problem.

Each time a job is selected the remaining positions of the sequence are completed with the non-selected jobs in initial sequence order and this becomes the current sequence. The timetabling procedure is used to insert unforced idle time and if the objective value is lower than the incumbent the incumbent is updated and the solution is retained. The pseudo-code for these hybrid procedures is provided below. In the code O_B is the objective of the best solution found, O_C, the objective of the current solution, and U is the set of unscheduled jobs.

Hybrid Procedure Pseudo-code

Step 1. Initialization:

- 1.1 Create an initial sequence by using either of the EDD, MDD or FV heuristics. This becomes the initial current sequence and the best sequence.
- 1.2 Create a schedule for the sequence by first calculating completion times without unforced idle time and then insert unforced idle time on the final machine as necessary to minimize total earliness and tardiness. Set $O_B =$ total earliness and tardiness found. Set $t_M =$ the start time, on the last machine, of the first job in the sequence (current time).
- 1.3 Set $k = 1$.
- 1.4 Set U to consist of the set of jobs to be scheduled.

Step 2. While $k \leq n$ do

- 2.1 Calculate the estimated makespan and slack threshold
- 2.2 For $j \in U$ calculate the priority index for job j.
- 2.3 Select *[k]* by picking the candidate job with the largest priority index.
- 2.4 Remove job *[k]* from set U.
- 2.5 Complete the sequence with the jobs in the set U in initial sequence order. This becomes the current sequence.
- 2.6 Create a schedule by calculating the completion times without unforced idle time and then inserting unforced idle time on the final machine as necessary to minimize total earliness and tardiness. Set $O_C =$ the resulting earliness and tardiness. If $O_C < O_B$ then set $O_B = O_C$ and retain the current sequence as the best sequence.
- 2.7 Set $k = k + 1$.
- 2.8 If $k < n$ then set $t_M =$ the start time of the job *kth* position of the current sequence starts processing on the last machine (current time).

Step 3. Stop. Return the best sequence.

Step 1 initializes the procedure by creating an initial sequence using either of the EDD, MDD or FV heuristics and then scheduling the sequence by inserting unforced idle time as needed to obtain an initial objective value and an estimated start time on the

last machine for the first job in a sequence. Step 2 is performed k times and each time it is performed it selects the job for position k, creates a candidate sequence by sequencing the remaining jobs in initial sequence order, and then schedules the sequence and calculates its associated objective value. If the objective value is better than the best objective found then the best objective is updated and the sequence is retained. This step also finds the estimated start time on the last machine for the job to be sequenced in position $k+1$. Step 3 terminates the procedure, updates the objective and outputs the best sequence found.

The following nine procedures are created. The FVEDD, FVMDD and FVFV procedures use the indexes used in the FV procedure during the second pass of the procedure. FVEDD uses the EDD rule to create the initial sequence, FVMDD, uses the MDD procedure, and FVFV uses the FV procedure. The LIN1EDD, LIN1MDD and LIN1FV procedures use the LIN1 index during the second pass and the procedures associated with their suffixes to create the initial sequences and similarly the LIN2EDD, LIN2MDD and LIN2FV procedures use the LIN2 index when the second pass is conducted.

We also created four other sets of procedures which use indexes in which $P_j\ (S)$ is redefined. The idea that motivates the creation of these four sets of procedures is that if unforced idle time is used and processing on the final machine is pushed later for a lot of jobs so that the jobs' processing on the final machine starts later than the processing on the next to final machine ends then the problem resembles the single-machine early/tardy problem with inserted idle time allowed and an adjustment in the index should occur. The adjustment in all four sets of procedures is that the index puts more weight on what a job's characteristics are on the last machine when unforced idle time has been used to force processing on the last machine to begin after processing has concluded on the next to last machine. In each of the procedures we define $P_j\ (S)$ based on whether the current estimated start time on the last machine (t_M) for the next job to be scheduled is greater than the finish time of the last job scheduled on the next to last machine ($t_M > C_{[k-1]M-1}\ (S)$).

The first two sets of procedures are referred to as H1 and H2 sets of procedures. In these two sets of procedures if $t_M > C_{[k-1]M-1}\ (S)$ then $P_j\ (S) = p_{jM}$ else $P_j\ (S) = C_{jM}\ (S) - C_{[k-1]1}\ (S)$. We then use the branches in the LIN1 index to select the job that will be in the *kth* position when using the H1 sets of procedures and we use the branches in the LIN2 index for the selection of the job in the *kth* position when using the H2 sets of procedures. In each set there are three procedures. The H1 procedures are H1EDD, H1MDD and H1FV and the H2 procedures are H2EDD, H2MDD and H2FV. The suffix for each procedure indicates how the initial sequence was created.

The last two sets of procedures are referred to as H3 and H4 sets of procedures. To help develop these procedures let $FT_{[j]M-1}\ (S)$ be the time the job in position j finishes its processing on machine $M-1$. Let $ST_{[j]M}\ (S)$ be the time the job in position j starts processing on machine M. Each time a current sequence is created and unforced idle time is inserted to create a solution we check to see how many jobs have an $ST_{[j]M}\ (S) > FT_{[j]M-1}\ (S)$ among the unselected jobs that are sequenced in initial sequence order. We use ND to refer to this value and then calculate a fraction $NDF = ND/(n-k)$. This fraction is used to help calculate $P_j\ (S)$ in this procedure each time a job is to be selected.

We use this fraction as a measure of the impact unforced idle time has on the future jobs. If the fraction is large then we expect most of the jobs to have their processing on the final machine start later than their processing on the next to last machine finishes and therefore the characteristics of a job that pertain to the final machine become more important than those of machines 1 through $m-1$ so the problem has a greater resemblance to the single-machine early/tardy problem with inserted idle time allowed as the fraction increases. The index in these procedures is a hybrid. We define $P_j\ (S)$ based on elements from the FV and LIN1 and LIN2 procedures. If $t_M > C_{[k-1]M-1}\ (S)$ then $P_j\ (S) = NDF^{*}p_{jM} + (1 - NDF)^{*}IT_j$ else $P_j\ (S) = 0.5^{*}IT_j + 0.5^{*}(C_{jM}\ (S) - C_{[k-1]M}\ (S))$. To select the job that will be in the *kth* position we use the branches in the LIN1 index for the H3 procedures and use the branches in the LIN2 index for the H4 procedures. The H3 procedures are H3EDD, H3MDD and H3FV. The H4 procedures are H4EDD, H4MDD and H4FV. The suffix for each procedure represents how the initial sequence was created.

We use the data in the example problem shown in Table 1 to demonstrate the procedure for H4MDD. We set the parameter $v = 0.20$ to calculate the slack threshold (slk_thr). The initial sequence using the MDD rule is: 3-1-5-4-2 and after inserting unforced idle time the total earliness and tardiness is 81. Table 14 shows the slack (s_j), time required (P_j) and the priority index for each job during each iteration of the procedure as well as the current time (t_M), NDF value, Makespan estimate (Mi), slack threshold (slk_thr), current sequence (sequence) and the current objective value (O_c). The final sequence is 1-5-3-4-2 which results in a total earliness and tardiness of 52.

6. Computational test of the multiple sequence dispatching heuristics and results

The multiple sequence dispatching procedures were also tested on the same data sets described in Section 4.1. To measure the performance of these procedures the results were compared against the results of the FV procedure. The reason for selecting this procedure was that it was generally the best performing procedure. The measure of performance % Dev vs FV is calculated as $\%\ Dev\ vs\ FV = [(Z_h - Z_{FV})/ Z_{FV}] * 100$, where Z_{FV} = the earliness and tardiness of the solution generated by FV procedure, and Z_h = the total earliness and tardiness of the solutions generated by the dispatching heuristic procedures with the improvement procedure applied. Since these heuristics require more processing time than the dispatching heuristics we also measured the processing time needed for each instance by each procedure.

Using the second set of instances, described in section 4.1, the parameter w, used to calculate the slack threshold used in the second pass of the LIN1, LIN2, H1, H2, H3 and H4 based procedures, and the parameters a, b and c used in the second pass of the three FV based procedures (FVEDD, FVMDD and FVFV) were determined. The w selected for LIN1EDD, LIN1MDD, LIN1FV, LIN2EDD, LIN2MDD, LIN2FV, H2EDD, H2MDD, H3EDD, H3MDD, H3FV, H4EDD and H4MDD procedures was 0.20, for the H1EDD, H1MDD and H1FV procedures was 0.10, for the H2FV procedure was 0.50 and for the H4FV procedure was 0.70. The b and c values selected for the FVEDD, FVMDD and FVFV procedures were $b = 0.70$ and $c = 50$. The values for a were: FVEDD, 0.80, FVMDD, 0.70 and FVFV, 060.

Tables 15, 16 and 17 show the *% Dev* vs *FV* for each procedure for each level of number of jobs to be sequenced (n) as well as the averages across all the levels of jobs. Table 15 shows the results for $M = 5$, table 16, for $M = 10$ and table 17, for $M = 20$.

The results show that the H4MDD procedure performed the best on average for the 5 and 10 levels of number of machines and H4FV was best on average when $M = 20$. The H4MDD procedure had the fourth best average performance for $M = 20$ and H4FV was sixth best on average for $M = 5$ and was fourth, for $M = 10$. Three procedures ranked in the top six procedures across all three levels for numbers of machines: H4MDD, H2FV and H4FV. Three other procedures, LIN1FV, LIN2FV, and H4EDD ranked in the top 10 for all three levels of numbers of machines. These results indicate that the H4 procedures were the most consistent and generally performed better. Also, the other three procedures started with the FV sequence which may not be that surprising, since the FV procedure

Table 14
Priority indexes using the H4MDD procedure for the example problem.

s_j/P_j/index	Iteration				
Job	1	2	3	4	5
1	18/19/0.95				
2	27/15/-1.80	8/15/-4.99	-6/15/0.07	-8/15/0.07	-19/15/0.07
3	33/2/-16.50	14/2/-6.91	0/2/0.50		
4	32/11/-2.91	13/11/-1.16	-1/11/0.09	-3/11/0.09	
5	21/19/-1.11	2/19/-0.06			
t_M	42	61	75	77	88
NDF	1.00	1.00	1.00	1.00	1.00
M_i	80	94	94	94	94
slk_thr	16.00	17.20	13.40	12.60	9.20
Selected job	1	5	3	4	2
sequence	1-3-5-4-2	1-5-3-4-2	1-5-3-4-2	1-5-3-4-2	1-5-3-4-2
Objective ($O_{c)}$	64	52	52	52	52

Table 15
% DEV vs. FV for the multiple sequence dispatching heuristics for $M = 5$.

Procedure	*n*								
	15	20	25	30	40	50	75	100	Ave.
LIN1EDD	-11.81	-7.90	-11.85	-10.36	-12.13	-10.60	-14.02	-13.32	-11.50
LIN1MDD	-13.06	-9.57	-12.41	-11.57	-13.36	-11.56	-15.58	-14.45	-12.70
LIN1FV	-12.83	-11.96	-12.70	-11.42	-14.00	-12.63	-14.43	-14.55	-13.07
LIN2EDD	-11.28	-8.85	-10.89	-10.97	-13.73	-12.29	-15.33	-15.61	-12.37
LIN2MDD	-13.45	-10.36	-13.09	-13.38	-15.06	-13.15	-16.60	-16.66	-13.97
LIN2FV	-12.17	-12.62	-12.81	-11.79	-14.16	-13.98	-15.37	-16.70	-13.70
FVEDD	-11.17	-9.39	-10.01	-8.11	-7.28	-6.68	-6.83	-7.20	-8.33
FVMDD	-14.49	-11.45	-13.23	-10.68	-10.71	-9.02	-9.54	-9.06	-11.02
FVFV	-10.54	-8.94	-8.20	-6.57	-6.63	-6.80	-6.74	-7.04	-7.68
H1EDD	-9.17	-10.30	-8.46	-11.00	-11.85	-8.30	-8.43	-8.38	-9.49
H1MDD	-13.11	-11.28	-11.17	-13.02	-13.59	-9.82	-11.34	-10.99	-11.79
H1FV	-10.26	-11.48	-9.37	-11.41	-13.02	-10.47	-9.75	-9.53	-10.66
H2EDD	-12.02	-11.17	-12.72	-15.70	-18.07	-17.95	-22.09	-23.87	-16.60
H2MDD	-14.93	-13.06	-14.75	-16.76	-19.88	-18.63	-23.22	-24.21	-18.18
H2FV	-10.88	-12.95	-12.99	-14.23	-17.07	-18.90	-21.65	-23.26	-16.49
H3EDD	-14.65	-14.49	-11.74	-9.58	-9.75	-7.98	-10.89	-10.15	-11.15
H3MDD	-14.65	-13.24	-12.78	-11.66	-11.34	-10.26	-12.12	-12.42	-12.31
H3FV	-13.03	-12.98	-12.02	-9.66	-11.46	-9.74	-10.30	-10.47	-11.21
H4EDD	-13.36	-14.40	-13.98	-14.25	-18.37	-18.34	-22.33	-23.44	-17.31
H4MDD	-15.06	-14.48	-14.67	-16.59	-19.01	-19.06	-22.74	-24.56	-18.27
H4FV	-9.29	-11.63	-11.55	-12.99	-16.33	-17.42	-20.67	-22.44	-15.29

Table 16
% DEV vs. FV for the multiple sequence dispatching heuristics for $M = 10$.

Procedure	*N*								
	15	20	25	30	40	50	75	100	Ave.
LIN1EDD	-3.65	-5.35	-3.08	-0.80	-4.17	-4.51	-5.47	-6.09	-4.14
LIN1MDD	-6.42	-7.13	-2.82	-3.86	-6.50	-6.65	-6.68	-6.79	-5.86
LIN1FV	-11.11	-11.45	-9.97	-8.61	-11.56	-11.37	-10.39	-11.84	-10.79
LIN2EDD	-4.06	-4.85	-2.84	-0.74	-4.15	-6.14	-6.36	-6.34	-4.44
LIN2MDD	-6.74	-6.93	-3.61	-3.60	-6.78	-7.92	-7.28	-8.26	-6.39
LIN2FV	-10.46	-11.39	-10.20	-8.82	-11.06	-11.52	-11.37	-12.90	-10.97
FVEDD	-10.66	-10.97	-7.56	-5.67	-7.40	-7.32	-5.96	-6.19	-7.72
FVMDD	-12.27	-13.05	-10.06	-8.66	-10.19	-9.31	-7.33	-7.25	-9.77
FVFV	-9.37	-10.36	-7.98	-6.83	-7.11	-7.52	-5.63	-5.48	-7.54
H1EDD	-4.27	-4.56	-2.54	-1.25	-2.55	-3.41	-2.86	-1.71	-2.89
H1MDD	-5.29	-6.95	-3.25	-3.81	-5.13	-5.64	-4.67	-3.71	-4.81
H1FV	-8.51	-10.08	-9.29	-8.74	-9.00	-9.12	-7.73	-7.65	-8.77
H2EDD	-3.31	-6.69	-4.75	-3.56	-8.11	-9.85	-14.08	-15.03	-8.17
H2MDD	-6.43	-8.10	-5.32	-6.39	-10.13	-12.02	-14.46	-15.03	-9.74
H2FV	-8.25	-10.39	-9.53	-9.69	-13.22	-14.78	-16.79	-19.53	-12.77
H3EDD	-13.37	-13.91	-10.27	-8.94	-9.03	-8.56	-7.97	-8.40	-10.06
H3MDD	-14.13	-14.19	-10.30	-8.70	-9.28	-9.79	-8.66	-9.15	-10.53
H3FV	-13.51	-13.20	-10.97	-9.85	-9.40	-7.91	-9.05	-9.45	-10.42
H4EDD	-8.61	-11.87	-9.80	-9.31	-12.57	-15.31	-16.90	-20.43	-13.10
H4MDD	-12.28	-13.04	-10.95	-10.49	-12.74	-15.63	-17.03	-20.73	-14.11
H4FV	-9.09	-10.57	-10.06	-9.63	-11.43	-14.16	-15.94	-18.76	-12.46

Table 17
% DEV vs. FV for the multiple sequence dispatching heuristics for $M = 20$.

Procedure	N								
	15	20	25	30	40	50	75	100	Ave.
LIN1EDD	2.38	5.46	2.41	5.37	5.90	7.40	2.62	1.96	4.19
LIN1MDD	1.34	2.89	-0.70	2.99	3.93	5.12	1.36	0.22	2.14
LIN1FV	-7.21	-7.08	-8.14	-6.58	-5.93	-5.69	-8.84	-8.75	-7.28
LIN2EDD	3.41	6.52	2.76	5.26	5.52	7.30	2.30	1.96	4.38
LIN2MDD	1.76	2.71	-0.46	2.68	2.88	4.71	0.32	0.06	1.83
LIN2FV	-7.47	-6.53	-7.54	-6.74	-6.52	-5.33	-8.82	-8.86	-7.23
FVEDD	-6.44	-6.94	-8.60	-6.01	-5.05	-3.51	-4.04	-4.46	-5.63
FVMDD	-7.43	-7.69	-9.75	-7.48	-6.49	-4.44	-5.66	-6.29	-6.90
FVFV	-6.85	-7.44	-10.09	-7.17	-5.83	-4.84	-5.64	-4.94	-6.60
H1EDD	4.20	6.54	2.39	6.90	7.94	10.35	4.05	2.95	5.67
H1MDD	2.52	4.36	-0.42	3.58	4.61	6.70	2.49	0.89	3.09
H1FV	-6.11	-4.54	-7.09	-6.04	-4.96	-3.92	-7.14	-6.76	-5.82
H2EDD	3.64	6.51	3.20	5.57	4.13	3.87	-1.21	-4.66	2.63
H2MDD	1.83	3.16	-0.82	2.48	1.27	1.30	-2.50	-5.73	0.12
H2FV	-5.03	-4.30	-6.15	-6.06	-6.69	-7.34	-12.31	-13.37	-7.66
H3EDD	-3.57	-5.98	-8.18	-4.70	-6.07	-4.32	-6.15	-7.98	-5.87
H3MDD	-9.04	-9.34	-10.24	-7.34	-4.63	-3.35	-6.49	-7.19	-7.20
H3FV	-9.28	-8.87	-9.14	-8.46	-7.26	-6.41	-8.02	-8.26	-8.21
H4EDD	-0.88	-4.62	-5.78	-5.45	-6.07	-6.15	-10.48	-13.01	-6.56
H4MDD	-5.96	-6.73	-7.86	-5.80	-5.51	-4.95	-10.49	-12.60	-7.49
H4FV	-5.84	-6.77	-7.89	-7.45	-6.62	-7.33	-12.14	-13.40	-8.43

Table 18
% DEV vs. FV for the hybrid dispatching heuristics by r for $n = 50$ and $M = 10$.

r	Procedure					
	LIN1FV	LIN2FV	H4MDD	H4EDD	H4FV	H2FV
0.0	-18.28	-19.09	-26.64	-27.32	-24.40	-25.08
0.2	-12.06	-12.24	-17.03	-17.22	-13.33	-14.88
0.4	-3.77	-3.24	-3.21	-1.40	-4.73	-4.38

Table 19
% DEV vs. FV for the hybrid dispatching heuristics by R for $n = 50$ and $M = 10$.

R	Procedure					
	LIN1FV	LIN2FV	H4MDD	H4EDD	H4FV	H2FV
0.2	-9.55	-10.25	-16.78	-18.53	-16.07	-18.44
0.6	-11.64	-11.37	-14.64	-14.78	-15.38	-16.40
1.0	-12.92	-12.94	-15.46	-12.64	-11.02	-9.51

was generally the best performing of the procedures described in section 3. The H3 based procedures (H3EDD, H3MDD and H3FV) performed very well when the number of jobs was smaller, frequently having the best performance but their performance deteriorated relative to the other procedures as the number of jobs increased.

In order to show the effect of the due date range (R) and tardiness factor (r) on the results tables 18 and 19 are presented. The tables report results for the six procedures that ranked in the top ten for all three levels of numbers of machines. Table 18 shows the *% Dev* vs. *FV* by due date tardiness factor (r) for $n = 50$ and $M = 10$.

The results by due date tardiness factor (r) show that for each of the six procedures the results compared to the FV procedure was best for $r = 0.0$, followed by $r = 0.2$ and then $r = 0.4$. This is to be expected because the procedures are trying to anticipate the amount of unforced idle time that will be used in order to generate better solutions and more unforced idle time is used when r is low. The H4MDD and H4EDD procedures were the two best procedures, and were very close, for $r = 0.0$ and 0.2 but were not as good as the other procedures when $r = 0.4$ (H4FV was the best).

Table 20
Average seconds per instance for the multiple sequence dispatching heuristics for $M = 20$.

n	Procedure					
	LIN1FV	LIN2FV	H4MDD	H4EDD	H4FV	H2FV
15	0.034	0.013	0.032	0.014	0.039	0.032
20	0.039	0.024	0.040	0.022	0.054	0.041
25	0.061	0.036	0.060	0.030	0.068	0.059
30	0.068	0.049	0.064	0.043	0.054	0.072
40	0.116	0.089	0.102	0.084	0.124	0.108
50	0.147	0.161	0.143	0.153	0.159	0.143
75	0.424	0.403	0.404	0.383	0.460	0.409
100	0.829	0.802	0.836	0.793	0.913	0.840

Table 19 shows the *% Dev* vs. *FV* by due date range factor (R) for $n = 50$ and $M = 10$.

The results show that the performance of the LIN1FV and LIN2FV procedures improved as R increased and the performance of the H4EDD, H4FV and H2FV procedures worsened as R increased. The performance of the H4MDD was consistent across all the levels of R and resulted in the best overall performance and was best when $R = 1.0$.

We also checked to see if the differences in the mean % Dev for each pair of heuristics are significantly different by performing a repeated measures ANOVA test. There were interaction effects present for the number of jobs and machines. Therefore, the test was performed separately for each combination of M and n. The results showed that there is a significant effect regarding the heuristic used for all combinations of M and n (the hypothesis that all the heuristics perform similarly was rejected). Post-hoc tests were then conducted between each pair of heuristics, using the Bonferroni correction for the number of comparisons being performed. A 5% level of significance was used.

The statistical tests showed that the improved performance for the six procedures identified above (H4MDD, H2FV, H4FV, LIN1FV, LIN2FV, and H3EDD) as being consistently the best performer compared to the other procedures was statistically significant for a large percentage of the combinations of M and n.

Table 20 shows the average computational time per instance used by the six procedures that ranked in the top ten for all three

Table 21
Average seconds per instance for the multiple sequence dispatching heuristics by r and R for $n = 100$ and $M = 20$.

r	R	Procedure					
		LIN1FV	LIN2FV	H4MDD	H4EDD	H4FV	H2FV
0.0	0.2	1.37	1.32	1.24	1.23	1.44	1.31
0.0	0.6	1.33	1.34	1.23	1.28	1.38	1.34
0.0	1.0	0.79	0.75	0.78	0.74	0.85	0.77
0.2	0.2	0.97	0.97	0.94	0.88	0.97	0.94
0.2	0.6	0.91	0.88	0.89	0.82	0.94	0.89
0.2	1.0	0.65	0.62	0.65	0.60	0.75	0.62
0.4	0.2	0.69	0.67	0.69	0.64	0.73	0.65
0.4	0.6	0.56	0.52	0.53	0.52	0.63	0.58
0.4	1.0	0.73	0.42	0.44	0.43	0.53	0.46

levels of numbers of machines by each level of number of jobs for $M = 20$. The amount of computational time per instance did not vary greatly by the level of number of machines so we did not include these results for $M = 5$ or 10.

These results show that the computational time required does not vary much based on which procedure is used. The H4EDD procedure required the least time for seven of the eight levels of number of jobs (the exception was $n = 50$) but the difference between the amount of time this procedure used and the amount of time used by the other procedures was small. The computational time required increased as the number of jobs was increased. All of the procedures required less than one second per instance for all levels of number of jobs. These results show that the procedures can solve large sized instances in a reasonable amount of time.

Table 21 shows the average computational time per instance used by the six procedures shown in the previous table for each combination of r and R for $n = 100$ and $M = 20$.

The results of this table show that as r increases the procedures generally need less computational time to generate a solution. The reason for this is that as due dates become tighter less unforced idle time is needed so the computational time needed to insert unforced idle time becomes lower. Also, as R increases the time required by the procedures to generate a solution is generally lower.

7. Conclusion

In this paper, we test seven dispatching heuristics for minimizing total earliness and tardiness in permutation flow shops when using unforced inserted idle time to reduce job earliness is allowed. Twenty one additional multiple sequence dispatching procedures were also developed. These procedures were tested on instances of various sizes in terms of the number of jobs and machines, and nine sets of distributions that determine the tightness and range of due dates. All of the procedures are very efficient and were able to generate solutions quickly for all the problem sizes tested.

The results show that using the multiple sequence procedures with the second pass generates solutions with lower total earliness and tardiness. The reason these procedures generated better solutions is that they estimate the amount of unforced idle time that will be used when selecting the job to be sequenced in a given position. The results also showed that the H4MDD and H4FV procedures were generally the best.

There are two areas for conducting additional research for the problem. One would be the development of metaheuristics for the problem. The second would be the development of an exact algorithm, such as a branch-and-bound algorithm, or a lower bounding method, for the problem, so that an estimate of how far the solutions generated by heuristic procedures deviate from the optimal objective function values (or a lower bound on those optimal values).

References

Baker, K. R. (1974). *Introduction to sequencing and scheduling*. New York: Wiley.

Baker, K. R., & Bertrand, J. W. M. (1982). A dynamic priority rule for scheduling against due-dates. *Journal of Operations Management, 3*(1), 37–42.

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

Chandra,, C., Mehta, P., & Tirupati, D. (2009). Permutation flow shop scheduling with earliness and tardiness penalties. *International Journal of Production Research, 47*, 5591–5610.

Davis, J. S., & Kanet, J. J. (1993). Single-machine scheduling with Early and Tardy Completion Costs. *Naval Research Logistics, 40*, 85–101.

Fernandez-Viagas, V., Dios, M., & Framinan, J. M. (2016). Efficient constructive and composit heuristics for the permutation flowshop to minimize total earliness and tardiness. *Computers and Operations Research, 70*, 38–68.

Fry, T. D., Armstrong, R. D., & Blackstone, J. H. (1987). Minimizing weighted absolute deviation in single machine scheduling. *IIE Transactions, 9*, 445–450.

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

Jackson, J. R. (1955). Scheduling a production line to minimize maximum tardiness. *Management Science Research Project*. Los Angeles: University of California.

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

Kim, Y. D., & Yano, C. A. (1994). Minimizing mean tardiness and earliness in single–machine scheduling problems with unequal due dates. *Naval Research Logistics, 41*, 913–933.

Madhushini, N., Rajendran, C., & Deepa, Y. (2009). Branch-and-bound algorithms for scheduling in permutation flowshops to minimize the sum of weighted flowtime/sum of weighted tardiness/sum of weighted flowtime and weighted tardiness/sum of weighted flowtime, weighted tardiness and weighted earliness of jobs. *Journal of the Operational Research Society, 40*, 991–1004.

Moslehi, G., Mirzaee, M., Vasei, M., & Azaron, A. (2009). Two-machine flow shop scheduling to minimize the sum of maximum earliness and tardiness. *International Journal of Production Economics, 122*, 763–773.

M'Hallah, R. (2014). An iterated local search variable neighborhood descent hybrid heuristic for the total earliness tardiness permutation flow shop. *International Journal of Production Research, 52*(13), 3802–3819.

Ow, P. S., & Morton, T. E. (1989). The Single-Machine Early Tardy Problem. *Management Science, 35*(2), 177–191.

Panwalkar, S. S., & Iskander, W. (1977). Survey of Scheduling Rules. *Operations Research, 25*(1), 45–61.

Rajendran, C. (1999). Formulations and heuristics for scheduling in a kanban flowshop to minimize the sum of weighted flowtime, weighted tardiness and weighted earliness of containers. *International Journal of Production Research, 37*, 1137–1158.

Schaller, J. E., & Valente, J. M. (2013a). An evaluation of heuristics for scheduling a non-delay permutation flow shop with family setups to minimize total earliness and tardiness. *Journal of the Operational Research Society, 64*(6), 805.

Schaller, J. E., & Valente, J. M. (2013b). A comparison of metaheuristic procedures to schedule jobs in a permutation flow shop to minimise total earliness and tardiness. *The International Journal of Production Research, 51*(3), 772.

Smith, W. E. (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly, 3*, 59–66.

Szwarc, W. (1993). Adjacent ordering in single-machine scheduling with earliness and tardiness penalties. *Naval Research Logistics, 40*(2), 229–244.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research, 64*, 278–285.

Valente, J. M. S. (2007). Dispatching heuristics for the single machine early/tardy scheduling problem with job-independent penalties. *Computers & Industrial Engineering, 52*(4), 434–447.

Valente, J. M. S. (2009). Beam search heuristics for the single machine scheduling problem with linear earliness and quadratic tardiness costs. *Asia-Pacific Journal of Operational Research, 26*, 319–339.

Valente, J. M. S., & Alves, R. A. F. S. (2005). Improved heuristics for the early/tardy scheduling problem with no idle time. *Computers & Operations Research, 32*(3), 557–569.

Vepsalainen, A. P. J., & Morton, T. E. (1987). Priority Rules for Job Shops with Weighted Tardiness Costs. *Management Science, 33*(8), 1035–1047.

Yano, C. A., & Kim, Y-D. (1991). Algorithms for a class of single machine weighted tardiness and earliness problems. *European Journal of Operational Research, 52*, 161–178.

Zegordi, S. H., Itoh, K., & Enkawa, T. (1997). A knowledgeable simulated annealing scheme for the early/tardy flow shop scheduling problem. *International Journal of Production Research, 33*, 1449–1466.