Int. J. Production Economics 224 (2020) 107542

Contents lists available at ScienceDirect

International Journal of Production Economics

journal homepage: http://www.elsevier.com/locate/ijpe

Minimizing total earliness and tardiness in a nowait flow shop

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

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

ARTICLE INFO

Keywords:
Scheduling
Heuristics
No-wait flow shop
Earliness and tardiness

ABSTRACT

This paper considers the problem of scheduling jobs in a no-wait flow shop with the objective of minimizing total earliness and tardiness. An exact branch-and-bound algorithm is developed for the problem. Several dispatching heuristics used previously for other environments and two new heuristics were tested under a variety of conditions. It was found that one of the new heuristics consistently performed well compared to the others. An insertion search improvement procedure with speed up methods based on the structure of the problem was proposed and was found to deliver much improved solutions in a reasonable amount of time.

1. Introduction

Scheduling problems with the objective of minimizing total earliness and tardiness have gained increased attention during the past four decades. A key reason for this is the adaption of supply chain management in which customers and suppliers have tried to have better coordination in their operations. This causes the early and tardy delivery of products to be viewed as poor quality service. Early deliveries result in unnecessary inventory that requires space, cash and resources needed to maintain and manage the inventory. Lost sales and the loss of customer good will are penalties that are the result of tardy delivery of products. This paper addresses this trend by considering an objective that sums the penalties for earliness and tardiness for a set of jobs to be processed in a no-wait flow shop.

A flow shop is a production shop that consists of two or more machines. In a flow shop each of the jobs to be processed uses the machines in the same order. In a no-wait flow shop once a job starts processing it must continue through the flow shop without any intermediate waiting. In some cases, the nature of the product is such that the no-wait restriction is a requirement. In many cases, organizations are adopting a lean production philosophy and want to minimize waste by not having any waiting between the machines. When jobs do not have to wait between the machines there is no in-process inventory and space is not wasted.

The objective in this problem is non-regular, therefore the insertion of idle time into a schedule could help to reduce the earliness of some jobs and thus improve the objective. In a traditional flow shop, there could be idle time on any of the machines but if additional idle time (unforced idle time) was to be used in the no-wait flow shop environment it would have to occur on the first machine because the storage of jobs is not allowed in between the intermediate stages. There are production environments, however, where the insertion of unforced idle time may not be productive. Korman (1994) and Landis (1993) provide specific examples of the undesirability of unforced idle time. If the capacity of the shop is limited relative to its production requirements, then unforced idle time should be avoided. If the machines are expensive to operate and starting the machines up again after the idle time is expensive then idle time should also be avoided. We do not consider the use of unforced idle time when developing sequences and schedules in this research.

2. Literature review

Research that is relevant to our problem is scheduling with earliness and tardiness penalties, scheduling flow shops with earliness and tardiness penalties and scheduling in a no-wait flow shop with these penalties. Many papers have been published that consider earliness and tardiness penalties. Baker and Scudder (1990) cover the early papers with these penalties. More recent research for early/tardy penalties was covered by Hoogeveen (2005). Most of the research for the early/tardy objective was for the single machine environment. Valente (2009) summarized more recent research for single machine environments with no idle time and an early/tardy objective. Kanet and Sridharan (2000) provide a review for the model that considers inserted idle time. One of the important considerations is timetabling algorithms to insert idle time to optimize a sequence for the single-machine problem. Papers that

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

https://doi.org/10.1016/j.ijpe.2019.107542
Received 12 December 2018; Received in revised form 22 October 2019; Accepted 27 October 2019
Available online 31 October 2019

address this topic include Fry et al. (1987), Davis and Kanet (1993) and Yano and Kim (1991). Davis and Kanet (1993), Szwarc (1993), Kim and Yano (1994), and Schaller (2007) consider the problem of finding an optimal sequence for the single-machine problem when considering inserted idle time.

We found ten papers that consider earliness and tardiness costs for flow shop environments without the no-wait restriction. Two early papers that considered the objective were Zegordi et al. (1995) and Rajendran (1999). Moslehi et al. (2009) considered the objective of minimizing the sum of maximum earliness and tardiness in a two-machine flow shop. Chandra et al. (2009) consider the flow shop problem with earliness and tardiness penalties when there is a common due date. Madhushini et al. (2009) developed branch-and-bound algorithms by for a variety of objectives including minimizing earliness and tardiness. Schaller and Valente (2013b) developed a genetic algorithm, M'Hallah (2014) developed a variable neighborhood search heuristic, and a constructive heuristic, as well as, local searches were developed by Fernandez-Viagas et al. (2016) for the problem. Family setups were incorporated into the problem by Schaller and Valente (2013a). Schaller and Valente (2019) consider the problem of scheduling a permutation flow shop to minimize total earliness and tardiness when unforced idle time is allowed and develop several heuristics for the problem.

To the best of our knowledge there have been no papers that have addressed the problem of minimizing total earliness and tardiness in a no-wait flow shop. However, there has been a great deal of research on scheduling problems in the no-wait flow shop environment. A review of early work was provided by Hall and Sriskandarajah (1996). Much of the research dealt with objectives that were measures of efficiency such as minimizing makespan or flowtime. Recent research with an objective of minimizing makespan includes Pan et al. (2008) and Laha and Chakraborty (2009). For the flowtime objective, Framinan et al. (2010) and Gao et al. (2011) are recent examples. There has been increased research in recent years on objectives that measure the shop's ability to meet due dates and include tardiness in the objective. Minimizing total tardiness is similar to our objective but does not penalize the early completion of jobs. Aldowaisan and Allahverdi (2012), Liu et al. (2013) and Ding et al. (2015) are recent examples of research for minimizing total tardiness in no-wait flow shops.

The next section provides a formal statement of the problem. In section four, an exact branch-and-bound algorithm is developed. Proposed dispatching heuristics are described in section five. In section six, computational experiments for the branch-and-bound algorithm and for the dispatching heuristics are described and the results are presented. In section seven, insertion search improvement procedures are presented, as well as a computational test of these procedures. Section eight concludes the paper.

3. Problem statement

We consider the problem where n jobs are processed by m machines in a no-wait flow shop. We use d_j to denote the distinct due date of job j ($j = 1, \ldots, n$). The processing time and completion time of job j ($j = 1, \ldots, n$) on machine i ($i = 1, \ldots, m$) are denoted by p_{ji} and C_{ji} respectively. We use E_j to denote the earliness of job j ($E_j = \max\{d_j - C_{jm}, 0\}$, for $j = 1, \ldots, n$) and T_j to denote the tardiness of job j ($T_j = \max\{C_{jm} - d_j, 0\}$, for $j = 1, \ldots, n$). The objective is to minimize $Z = \sum_{j=1}^{n} E_j + T_j$.

In this paper we only consider no-wait flow shops. A permutation flow shop is a flow shop in which the order the jobs are processed remains the same on each machine. Therefore no-wait flow shops are also permutation flow shops so a solution is defined by determining the order in which the jobs will be processed. Also, for each pair of jobs it can be determined how much idle time is needed on the first machine between the finish time of the job sequenced first and the start time of the job sequenced second. This time is the same regardless of where the pair of jobs appears in the sequence so it can be calculated before sequencing jobs. We refer to this time as the delay or offset time and use OS_{jk} to denote the offset time needed on the first machine if job j is sequenced immediately after job k for job j not to wait at any of the machines. A dummy job 0 is used to take into consideration that a job may be first in a sequence and $OS_{j0} = 0$ for $j = 1, \ldots, n$. Once a start time for a job is determined, its completion time on the final machine is easy to determine as we only need to add the sum of the job's processing times to its start time. Let *[j]* represent the job that is in the *j*th position of a given sequence. The completion time of the job in position j of the sequence $C_{[j]m} = C_{[j-1]1} + OS_{[j][j-1]} + \sum_{i=1}^{m} p_{[j]i}$, where $C_{[0]1} = 0$.

To calculate the offset time between a pair of jobs j and k when job j is to immediately follow job k (OS_{jk}) we can use the following.

$$OS_{jk} = \max_{i=1,\ldots,m-1} \left\{ \sum_{l=1}^{i} p_{kl+1} - \sum_{l=1}^{i} p_{jl} \right\}.$$

The offset times between pairs of jobs can be thought of as a sequence-dependent setup time on the first machine. Using this concept, the problem can be transformed into a single-machine early/tardy problem with sequence-dependent setups. Let d_j' be a modified due date for job j by setting $d_j' = d_j - \sum_{i=2}^{m} p_{ji}$. This is the time job j must finish on the first machine in order to finish on time on the last machine since the no-wait constraint means that the job is processed on all machines without any idle time between machines. The earliness of job j, E_j, can be equivalently defined as: $E_j = \max\{d_j' - C_{j1}, 0\}$, for $j = 1, \ldots, n$ and the tardiness of job j, T_j, is defined as: $T_j = \max\{C_{j1} - d_j', 0\}$, for $j = 1, \ldots, n$. The problem then becomes:

$$\text{Min } Z = \sum_{j=1}^{n} E_{[j]} + T_{[j]} \tag{1}$$

Subject to:

$$C_{[j]1} = C_{[j-1]1} + OS_{[j][j-1]} + p_{[j]1}. \text{ For } j = 1, \ldots, n \tag{2}$$

We use this formulation to help us develop an exact branch-and-bound algorithm in the next section.

4. An exact branch-and-bound algorithm

In this section we develop an exact branch-and-bound algorithm. This algorithm will allow us to obtain an optimal solution for the problem, at least for small problems, which will help us evaluate the quality of solutions generated by heuristic approaches. In this branch-and-bound algorithm, a node in the branch-and-bound tree represents an initial partial sequence of jobs. For each node in the branch-and-bound tree a lower bound on the optimal objective value is calculated and conditions are examined that could help fathom the node. An incumbent value, that represents the value of the total earliness and tardiness of the current best sequence, is compared to the lower bound found for a node and if the incumbent value is less than or equal to the lower bound the node is fathomed. If a complete sequence is found with an objective value that is less than the incumbent's value, then the incumbent is updated and retained as the best sequence found. Initially the incumbent value is set equal to M, where M is a very large value.

4.1. Lower bounds

In this section we consider how to obtain a lower bound of the total earliness and tardiness for the completion of an initial partial sequence. For a given initial partial sequence we would not know the completion times on the first machine for the unscheduled jobs or what position of the completed sequence their modified due dates would be in. In order to develop a lower bound, we use modified due dates and completion times for the remaining positions in the sequence that will provide a

lower bound on the objective. We first show how modified due dates can be obtained that when substituted for the actual modified due dates results in a lower bound on the objective and then describe two approaches for obtaining lower and upper bounds on completion times on the first machine for the jobs in the positions of the sequence that have not yet been scheduled. By comparing these lower and upper bounds for completion times to the substituted modified due dates a lower bound on the objective is obtained.

For the objective of minimizing total earliness and tardiness on a single machine Schaller (2007) proved that if the due dates in EDD order are substituted for the actual due dates and compared to the actual completion times a lower bound on the total earliness and tardiness is obtained. For our problem let d'_{EDD} be the modified due dates sorted in non-descending order. We call these due dates EDD modified due dates. Then, $\sum_{j=1}^{n}(\max\{C_{[j]1} - d'_{EDD[j]}, 0\}) + \max\{d'_{EDD[j]} - C_{[j]1}, 0\}) \leq \sum_{j=1}^{n}(\max\{C_{[j]1} - d'_j, 0\} + \max\{d'_j - C_{[j]1}, 0\})$. This result can be used as a substitute for (1) to provide a lower bound. We also would not know the actual completion times ($C_{[j]1}$ for $j = 1, \ldots, n$) of the jobs in a sequence but we can calculate lower and upper bounds on these completion times. Let $LBC_{[j]1}$ be a lower bound on the completion time on the first machine of the job in position j of the sequence and let $UBC_{[j]1}$ be an upper bound on the completion time on the first machine of the job in position j of the sequence. Let σ represent an initial partial sequence with p jobs. We can use the following mathematical program to obtain a lower bound for total earliness and tardiness on the completion of the partial sequence.

$$\text{Min } Z_{LB} = \sum_{j=1}^{p}\left(\max\left\{C_{[j]1} - d'_{[j]}, 0\right\} + \max\left\{d'_{[j]} - C_{[j]1},\ 0\right\}\right) + \sum_{j=p+1}^{n}\left(\max\left\{C_{[j]1} - 0d'_{EDD[j]},\ 0\right\} + \max\left\{d'_{[j]} - C_{EDD[j]1}, 0\right\}\right) \quad (3)$$

Subject to:

$$C_{[j]1} = C_{[j-1]1} + OS_{[j][j-1]} + p_{[j]1}.\ \text{For } j = 1, \ldots, p \quad (4)$$

$$C_{[j]1} \geq LBC_{[j]1}\ \text{For } j = p+1, \ldots, n \quad (5)$$

$$C_{[j]1} \leq UBC_{[j]1}\ \text{For } j = p+1, \ldots, n \quad (6)$$

In order to implement the above mathematical program, we need a method to develop lower and upper bounds on the completion times on the first machine for the jobs in positions $p+1$ through n for the completion of the partial sequence. We develop two approaches for obtaining these lower and upper bounds. For both approaches we let σ' be the set of jobs not in the initial partial sequence.

4.1.1. Approach 1

Using this approach, we look at the minimum offset times for each job and the processing times on the first machine sorted in shortest processing time order (SPT) to obtain lower bounds on completion times. We use the maximum offset times for each job and the processing times on the first machine sorted in longest processing time order (LPT) to obtain upper bounds on the completion times. Let $p_{SPT[j]1}$ be equal to the processing time on the first machine of the *jth* job in the set σ' if the jobs are sorted in SPT order and $p_{LPT[j]1}$ be equal to the processing time on the first machine of the *jth* job in the set σ' if the jobs are sorted in LPT order. Let $SOS_{\sigma'[p]}$ be the shortest offset time among the jobs in σ' if sequenced immediately after the last job sequenced in σ and $LOS_{\sigma'[p]}$ be the associated longest offset time among these jobs. Lower and upper bounds on the completion time on the first machine for the next job to be sequenced after σ are:

$$LBC_{[p+1]1} = C_{[p]1} + SOS_{\sigma'[p]} + p_{SPT[1]1} \text{ and } UBC_{[p+1]1} = C_{[p]1} + LOS_{\sigma'[p]} + p_{LPT[1]1}.$$

To obtain the lower and upper bounds for the completion times for positions $p+2$ through n we let $SOS_{j\sigma'}$ and $LOS_{j\sigma'}$ be the minimum and maximum offset time for job $j \in \sigma'$ among the jobs in the set σ'. We then sort these times by shortest and longest time, respectively. Let $OS_{SOS[j]\sigma'}$ be equal to the offset time for the job in the set σ' if the jobs are sorted in shortest minimum offset time order and $OS_{LOS[j]\sigma'}$ be equal to the offset time for the job in the set σ' if the jobs are sorted in longest maximum offset time order. The lower and upper bounds on the completion times on the first machine for the jobs that will fill positions $p+2$ through n are:

$$LBC_{[j]1} = LBC_{[j-1]1} + OS_{SOS[j-1]\sigma'} + p_{SPT[j]1} \text{ and } UBC_{[j]1} = UBC_{[j-1]1} + OS_{LOS[j-1]\sigma'} + p_{LPT[j]1} \text{ for } j = p+2, \ldots, n.$$

In the two equations above $n - p - 1$ shortest and longest minimum and maximum offset times are used (for each j a $j - 1$ offset time is used). The reason for this is in the first position $(p + 1)$ the shortest and longest offset times among the jobs in the set σ' if scheduled immediately after job [p] are used.

4.1.2. Approach 2

In this approach we also look at the minimum and maximum offset times for each job to develop lower and upper bounds on the completion times on the first machine but combine these values with each job's processing time on the first machine. By doing this we hope to achieve better bounds than the first approach. Let $POS_{j[p]}$ be equal to the offset time of job j if it is sequenced immediately after the job in position p plus job j's processing time on the first machine ($POS_{j[p]} = p_{j1} + OS_{jp}$ for $j \in \sigma'$). Let $MINPOS = \min\{POS_{j[p]}\}$ for $j \in \sigma'$ and $MAXPOS = \max\{POS_{j[p]}\}$ for $j \in \sigma$'.

Lower and upper bounds on the completion time on the first machine for the next job to be sequenced after σ are:

$$LBC_{[j]1} = C_{[p]1} + MINPOS \text{ and } UBC_{[j]1} = C_{[p]1} + MAXPOS.$$

To obtain the lower and upper bounds for the completion times for positions $p+2$ through n, let $SOS_{[j]\sigma'} = OS_{SOS[j]\sigma'} + p_{j1}$ and $LOS_{j\sigma'} = OS_{SOS[j]\sigma'} + p_{j1}$ for job $j \in \sigma$'. We then sort these times by shortest and longest time, respectively. Let $SOS_{S[j]}$ be the *jth* shortest *SOS* time and $LOS_{L[j]}$ be the *jth* longest *LOS* time. The lower and upper bounds, using this approach, on the completion times on the first machine for the jobs that will fill positions $p+2$ through n are:

$$LBC_{[j]1} = LBC_{[j-1]1} + SOS_{S[j-1]} \text{ and } UBC_{[j]1} = UBC_{[j-1]1} + LOS_{L[j-1]} \text{ for } j = p+2, \ldots, n.$$

Similar to the first approach, in the two equations above $n - p - 1$ shortest and longest minimum and maximum offset plus first machine processing times are used (for each j a $j - 1$ time is used). The reason for this is in the first position $(p + 1)$ the shortest and longest offset plus first machine processing times among the jobs in the set σ' if scheduled immediately after job *[p]* are used.

4.2. Dominance conditions

In this section we present conditions for jobs that are adjacent in a partial sequence that can eliminate further consideration of the partial sequence. Let S be a partial sequence with $p \geq 3$ jobs. Let job k be the last job in the partial sequence, jobs h and j are adjacent jobs that immediately precede job k with job h before job j, and job g immediately precedes job h in the partial sequence S. If there are only three jobs in the partial sequence, then job g is a dummy job 0 and any job immediately following has an offset time of 0. We consider a partial sequence S' that is the same as S except the positions of jobs h and j are exchanged so job h is after job j and immediately precedes job k and job j is the first job after job g. Let $C_{k1}(S)$ and $C_{k1}(S')$ be the completion times of job k on the first machine in schedules S and S' respectively. Let $T_h(S)$, $T_j(S)$, $T_k(S)$, $T_h(S')$, $T_j(S')$, $T_k(S')$ be the tardiness of jobs h, j, and k in partial sequences S and S' respectively and let $E_h(S)$, $E_j(S)$, $E_k(S)$, $E_h(S')$, $E_j(S')$, $E_k(S')$ be the earliness of jobs h, j, and k in partial sequences S and S' respectively.

Note that the completion times on the first machine, or the earliness or tardiness, will not change for the jobs before jobs h and j (job g and any jobs preceding g). Let C_{diff} be the absolute value of the change in the completion time of job k on the first machine if the partial sequence S' is used instead of S ($C_{diff} = |C_{k1}(S') - C_{k1}(S)| = |(OS_{jg} + OS_{hj} + OS_{kh}) - (OS_{hg} + OS_{jh} + OS_{kj})|$). Let OBJ_{diff} be the difference in the objective values of the two partial sequences. $OBJ_{diff} = (T_h(S) + T_j(S) + T_k(S) + E_h(S) + E_j(S) + E_k(S)) - (T_h(S') + T_j(S') + T_k(S') + E_h(S') + E_j(S') + E_k(S'))$. We have the following condition that can be used to eliminate partial sequences from further consideration.

Condition 1. If $OBJ_{diff} > C_{diff} * (n - p)$ then partial sequence S can be eliminated from further consideration.

Proof: Let σ' be the set of jobs that are not included in the partial sequences S and S'. There are $(n-p)$ jobs in σ'. For any completion of the partial sequences the jobs in σ' will have their completion times on the first machine change by C_{diff} if the initial partial sequence S' is used instead of S. Therefore the earliness or tardiness of each of these jobs would increase by at most (an upper bound on the increase) C_{diff} and $C_{diff} * (n-p)$ is an upper bound on the total increase in earliness and tardiness for the jobs in the set σ' if S' is used instead of S. OBJ_{diff} represents the savings in earliness and tardiness of the jobs in S' if it is used instead of S and if it is larger than $C_{diff} * (n-p)$ the initial partial sequence S' will result in a lower total earliness and tardiness when completed than the initial partial sequence S.//

We could strengthen the above condition if we knew how many tardy or early jobs would be in the set σ' under the two initial partial sequences. For example if $C_{k1}(S') < C_{k1}(S)$ and there are 3 jobs in the set σ' with a modified due date d_j' ($j \varepsilon \sigma'$) $\leq C_{k1}(S')$ then we know that using S' will decrease the tardiness of each of these jobs by C_{diff} so we can add these savings to OBJ_{diff} and compare it to $C_{diff} * (n-p-3)$ to determine whether or not to eliminate S. Let $C_{dec} = \max\{C_{k1}(S) - C_{k1}(S'), 0\}$ and $C_{inc} = \max\{C_{k1}(S') - C_{k1}(S), 0\}$. We also use the lower and upper bounds on the completion times on the first machine of the jobs in positions $p+1$ through n for the jobs in the set σ' in the initial partial sequence S as well as the modified due dates d' to find lower bounds on the number of tardy and early jobs. Let LB_{ntdy} and LB_{nely} be the lower bounds on the number of tardy and early in the set σ' respectively. If $C_{dec} > 0$ ($C_{k1}(S') < C_{k1}(S)$) we can also find a lower bound on the decrease in tardiness that will result if S' is used for jobs that will be tardy in S and an upper bound on any earliness these jobs would have if S' is used instead of S. We do the same thing for the minimum number of early jobs in the set σ' finding lower bounds on the decrease in earliness that will result if S' is used for jobs that will be early in S and an upper bound on any tardiness these jobs would have if S' is used instead of S. Based on this we update OBJ_{diff} and check the following two conditions.

Condition 2. If $C_{dec} > 0$ and $OBJ_{diff} > C_{dec} * (n - p - LB_{ntdy})$ then partial sequence S can be eliminated from further consideration.

Condition 3. If $C_{inc} > 0$ and $OBJ_{diff} > C_{inc} * (n - p - LB_{nely})$ then partial sequence S can be eliminated from further consideration.

5. Dispatching heuristics tested

Several heuristics were tested for the problem. Each of the heuristics tested are either variants of heuristics used in other environments or were motivated by heuristics used in other environments but were modified for the no-wait flow shop environment.

We use the following notation in the heuristics. S is a current partial schedule. If job j ($j \notin S$) is the next job scheduled after S, $C_{jm}(S)$ is job j's completion time. We use $s_j(S)$ to represent the slack of job j if job j is the next job scheduled after S, where $s_j(S) = d_j - C_{jm}(S)$. Additionally, let t be the current availability time of machine 1 under schedule S. This is the time the last job in schedule S completes its processing on the first machine. $P_j(S)$ ($P_j(S) = C_{jm}(S) - t$) is used to represent the time the shop is allocated to job j if job j is the next job scheduled after S, which includes the total processing time of job j plus the initial idle time on the first machine. Therefore, since the problem deals with a no-wait flow shop environment, $Pj(S)$ includes both the job's cumulative processing time and its offset time. Suppose k jobs have been selected and we are now selecting the $k+1$st job. Let $[k]$ represent the job in position k. $P_j(S) = OS_{j[k]} + \sum_{i=1}^{m} p_{ji}$.

Table 1
Simple dispatching heuristics.

Heuristic	Reference (s)	Priority index calculation
EDD (earliest due date rule)	Jackson (1955)	d_j
MDD (modified due date rule)	Baker and Bertrand (1982), Vepsalainen and Morton (1987)	$MDD_j(S) = \max\{d_j, C_{jm}(S)\}$
SLK (minimum slack rule)	Panwalkar and Iskander (1977), Vepsalainen and Morton (1987)	$s_j(S) = d_j - C_{jm}(S)$
SLK/P (minimum slack per required time rule)	Panwalkar and Iskander (1977), Vepsalainen and Morton (1987)	$SLK/P_j(S) = s_j(S)/P_j(S)$

5.1. Simple dispatching heuristics

We considered four simple dispatching heuristics. We include these heuristics because they are widely used in other environments, easily adopted to the no-wait flow shop environment when earliness and tardiness is the objective, and very efficient. These dispatching heuristics are listed in Table 1. For each heuristic, an abbreviation, as well as its name is given, references for its initial use, and the priority index used as well as its calculation. In each of the heuristics, at each iteration the job with the minimum priority index is selected to be appended to a partial sequence.

5.2. Dispatching rules with more advanced indexes

The dispatching rules in this section consider a variety of conditions to develop priority indexes that create the job sequence to be used. The indexes in several of these rules have multiple branches to allow for flexibility to respond to conditions.

These rules are based on rules that have been used in other environments (i.e. single-machine, permutation flow shop without the no-wait restriction, job shop) but are modified to consider characteristics that are specific to the no-wait flow shop. Using rules that are specifically tailed to an environment and the considered objective can be very important. For example Vinod and Sridharan (2011) and Xiong et al. (2017) developed dispatching rules that are tailed to the job shop environment and were shown to outperform simple rules such as the ones in the previous section.

5.2.1. LIN1 and LIN2 LIN-ET rules

Ow and Morton (1989) developed two rules for minimizing total weighted earliness and tardiness for a single machine. We use two rules that are based on Ow and Morton (1989)'s rules but are modified to reflect the no-wait flow shop environment. The resulting procedures, based on these rules, are denoted by LIN1 and LIN2. These rules, at each iteration, select the job to append to an initial partial sequence. The job selected is the one with the largest value of the associated priority index:

$$LIN1_j(S) = \begin{cases} \dfrac{1}{P_j(S)} & \text{if} \quad s_j(S) \leq 0 \\ \dfrac{1}{P_j(S)} - \dfrac{s_j(S)}{slk\_thr} \times \dfrac{2}{P_j(S)} & \text{if} \quad 0 < s_j(S) < slk\_thr \\ -\dfrac{1}{P_j(S)} & \text{if} \quad s_j(S) \geq slk\_thr \end{cases}$$

and

$$LIN2_j(S) = \begin{cases} \dfrac{1}{P_j(S)} & if \quad s_j(S) \le 0 \\ \dfrac{1}{P_j(S)} - s_j(S) \times \left(\dfrac{1}{slk\_thr \times P_j(S)} + \dfrac{1}{P_j(S)} \right) & if \quad 0 < s_j(S) < slk\_thr \\ -\dfrac{s_j(S)}{P_j(S)} & if \quad s_j(S) \ge slk\_thr \end{cases},$$

In the above indexes *slk_thr* is a parameter that is used to identify slack that is greater than a threshold value selected by the scheduler.

The above priority indexes represent three branches for both heuristics. When jobs are late or on time, the first branch is used and is similar to the shortest processing time rule used to minimize tardiness on a single-machine. Here the shortest time refers to a job's cumulative processing time plus the idle time on the first machine to ensure the job does not have to wait between machines.

In the third branch, the job is quite early because its slack is greater than the threshold. A job that satisfies the criteria for this branch would only be selected if all the unselected jobs satisfy the criteria. In this branch, LIN1 is using a longest time approach to select a job and LIN2 is using a minimum slack per required time approach to select a job. A linear interpolation between $s_j\ (S) = 0$ and $s_j\ (S) = slk\_thr$ is applied for the middle branch consistent with the LIN–ET procedure of Ow and Morton (1989).

The *slk_thr* parameter is calculated as follows. At each iteration, representing a partial schedule S with k jobs, 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 $0 \le w \le 1$ is a user-defined parameter. To calculate this lower bound we first obtain a lower bound on the completion of processing on the first machine of the unscheduled jobs and then add to this time the minimum cumulative processing time on machines two through m. To find the lower bound for completion on the first machine we sum the processing times on the first machine of the unscheduled jobs and add this to t ($t + \sum_{j \notin S} p_{[j]1}$). This lower bound is weak because the offset times of the unscheduled jobs is not included. To strengthen this lower bound we find the lowest offset time for each unscheduled job if it is not scheduled next (min$\{OS_{j \notin S, l \notin S}\}$ and sum the $n - k - 1$ of these times and add to this the minimum offset time among the unscheduled jobs if scheduled next. The lower bound on the offset times is then added to the lower bound above to obtain a lower bound on the completion time on the first machine. We then add the minimum among the unscheduled jobs of the cumulative processing time for machines two through m to obtain a lower bound on the makespan.

5.2.2. A heuristic based on Fernandez-Viagas et al. (2016)'s constructive heuristic

A constructive heuristic for permutation flow shops when waiting is allowed between machines was developed by Fernandez-Viagas et al. (2016). The heuristic adds one job at a time to a partial sequence based on an index. During each iteration of the procedure the problem is classified based on the due dates of the unselected jobs in order to determine the index to use to select the next job in the sequence. The index used is based on whether the due dates are relative tight, relatively loose or are neither relatively tight or loose. For details about the procedure see Fernandez-Viagas et al. (2016).

In the original Fernandez-Viagas et al. (2016) heuristic a variable is used to calculate the weighted idle time of the candidate jobs. The weighted idle variable in the Fernandez-Viagas et al. (2016) heuristic is based on the idle time that occurs on all the machines if a job is to be scheduled next. Since in our problem the shop is a no-wait flow shop and idle time only occurs on the first machine, we modify this heuristic by using the idle time on the first machine for each unscheduled job if scheduled next. Let IT_{jk} be the idle time on the first machine of the candidate jobs if scheduled next (in the $k + 1$st position): $IT_{jk} = OS_{j[k]}$. This redefined variable is then used in the Fernandez-Viagas et al. (2016) indexes to select a job during each iteration.

This procedure is referred to as FV in this paper.

5.2.3. Job shop dispatching heuristics modified for the No-Wait flow shop

It is very challenging to meet due dates in a job shop environment. Various dispatching heuristics have been tested and evaluated using due date based measures. Vinod and Sridharan (2011) found that two were effective for a variety of conditions, combination of critical ratio and shortest processing time (CRSPT) (Anderson and Nyirenda, 1990; Raghu and Rajendran, 1993), and combination of slack per remaining processing time and shortest processing time (SLRPT) (Anderson and Nyirenda, 1990; Raghu and Rajendran, 1993).

In a job shop environment, each operation must be sequenced. In our problem, since it is a no-wait flow shop, we only need to develop a single sequence for the jobs to be processed in the shop. In section 3 it was shown that the problem can be modeled as a single-machine problem with sequence-dependent setups. Therefore, we only use the rules to sequence the first machine (operation) in our implementation.

For the CRSPT rule, we first define the critical ratio (CR_j) for a job j. If k jobs have already been sequenced and we are selecting the job for the k + 1st position then $CR_j = (d_j - t)/(\sum_{i=1}^{m} p_{ji} + OS_{j[k]})$. For each job j that has not been sequenced we define the index $Z_j = \max\{CR_j * p_{j1}, p_{j1}\}$ and at each iteration select the job with the minimum Z_j value. We refer to this rule as CRSA in this paper. We created a second version of this rule, referred to as CRSB in this paper, which includes the offset time ($OS_{j[k]}$) in the SPT part of the rule. The index for this rule is $Z_j = \max\{CR_j * (p_{j1} + OS_{j[k]}), p_{j1} + OS_{j[k]}\}$.

For the SLRPT rule, we define the slack per remaining time (S/RPT_j) for each job j as: $S/RPT_j = (d_j - t - \sum_{i=1}^{m} p_{ji} - OS_{j[k]})/(\sum_{i=1}^{m} p_{ji} + OS_{j[k]})$. The index for this rule is $Z_j = \max\{S/RPT_j * p_{j1}, p_{j1}\}$ and at each iteration we select the job with the minimum Z_j value. We refer to this rule as SLRA in this paper. We also created a second version of this rule, referred to as SLRB in this paper, which includes the offset time ($OS_{j[k]}$) in the SPT part of the rule. The index for this rule is $Z_j = \max\{S/RPT_j * (p_{j1} + OS_{j[k]}), p_{j1} + OS_{j[k]}\}$.

5.2.4. Modified LIN-ET rules

Two new rules are developed that are very similar to the LIN1 and LIN2 rules. These new rules are referred to as H1 and H2. The H1 rule uses the LIN1 index to select a job for each position of the sequence. The H2 rule uses the LIN2 index to select a job for each position of the sequence. The difference between these new rules and the associated LIN rules is in how we calculate $P_j\ (S)$ for each unscheduled job. For both rules there are two changes in how $P_j\ (S)$ is calculated. The first is that we use the job's processing time on the first machine p_{j1} instead of the

Table 2
Average seconds used and number solved.

Number of jobs (n)		Procedure					
		BBD			BBD′		
		Number of Machines (m)			Number of Machines (m)		
		5	10	20	5	10	20
8	Seconds	0.081	0.119	0.167	0.107	0.096	0.109
	# Solved	90	90	90	90	90	90
10	Seconds	1.540	1.228	1.558	1.344	1.351	1.806
	# Solved	90	90	90	90	90	90
12	Seconds	22.67	27.90	26.85	24.80	31.46	33.88
	# Solved	90	90	90	90	90	90
15	Seconds	378.90	458.34	518.60	386.72	476.29	535.58
	# Solved	47	39	26	45	34	24

cumulative processing time. The second change is an adjustment based on a job's current offset time compared to its average offset time if it is not selected in the current iteration. Assume q jobs have been selected, job *[q]* was the last job selected and σ' is the set of $n-q$ unselected jobs. Let $\overline{OSj\sigma'} = \sum_{k\in\sigma' \& k\neq j} OS_{jk}/(n-q-1)$. Let $AD_j = OSj[k] - \overline{OSj\sigma'}$. We set $P_j(S) = OS_{j[q]} + p_{j1} + AD_j$. Since AD_j can be negative, if $P_j(S) < 1$ we set $P_j(S) = 1$. This allows us to not only consider a job's current offset time but also the difference between the current offset time and the offset time that could occur if the job is sequenced later.

6. Computational results of tests for the branch and bound procedure and dispatching heuristics

This section describes tests that were conducted to assess the efficiency of the branch-and-bound procedure and the accuracy of the dispatching procedures. First the tests and results for the branch-and-bound procedure are described and presented, and then the tests and results for the dispatching procedures are described and presented.

6.1. Computational results for the branch-and-bound procedure

The branch-and-bound procedure, described in section four, was tested on instances generated by Schaller and Valente (2019). The instances used in this test consist of four levels of the number of jobs, $n = 8$, 10, 12, and 15, three levels of numbers of machines, $m = 5$, 10, and 20 and nine combinations of due date range and tightness distributions. For each combination of the above levels, 10 instances were generated. To generate the processing times of each job on each machine a uniform distribution was used. These times were generated over the integers 1 and 100. Due dates were randomly generated for the jobs using a uniform distribution over the integers $MS(1-r-R/2)$ and $MS(1-r+R/2)$, where MS is an estimate for the makespan for the instance. The estimate is based on the Taillard (1993)'s lower bound. R and r, referred to as the due date range and tardiness factors, are parameters. The levels of due date range (R) tested are $R = 0.2$, 0.6 and 1.0 and the levels of due date tightness (r) tested are $r = 0.0$, 0.2 and 0.4.

Turbo Pascal was used to code the branch-and-bound procedure and the procedure was tested on a Dell Inspiron 1525 GHz Lap Top computer. In order to see the effect of the dominance conditions, two versions of the branch-and-bound procedure were created. One version of the procedure includes the dominance conditions, BBD and the other version does not include the dominance conditions, BBD'. The branch-and-bound procedures were performed for a maximum of 600 s for an instance. If a procedure was unable to prove an optimal solution for an instance within the time limit it was terminated. For each procedure and for each combination of n, m, R and r we recorded the average seconds used per instance and the number of instances solved within 600 s.

Table 2 shows the results for each level of number of jobs and number of machines. For each level of number of machines and jobs the average number of seconds used and the number of problems solved to optimality within 600 s are shown for each procedure.

The results show that as the number of jobs increases the time required to solve instances increases rapidly and when $n = 15$ less than half the instances were solved within 600 s (the procedures were able to solve all the instances with $n \leq 12$ within 600 s). There appears to be some increase in the time used by the procedures as the number of machines increases, particularly for $n = 15$. These results indicate that the branch-and-bound procedure can only solve small problems, in terms of numbers of jobs, in a reasonable amount of time. The results also show that including the dominance conditions generally helps the performance of the procedure. The version of the procedure with the dominance conditions used less time when $n \geq 12$ and was able to solve more problems within the 600 s time limit for each level of number of machines when $n = 15$.

We also looked at how the due date range and tardiness factors affected the results. The due date range factor had a larger effect than the due date tightness factor on both the time used and the number of instances solved. As the range of due dates increases the procedure uses less time and solves more problems, also as the due dates become tighter the procedure uses less time and solves more problems. As example, for $n = 15$ the BBD version of the procedure was able to solve 34.4% of the instances when $r = 0.0$, 42.2% when $r = 0.2$, and 47.9% when $r = 0.4$; for the due date range factors the BBD procedure was able to solve 8.9% of the instances when $R = 0.2$, 35.6% when $R = 0.6$, and 80.0% when $R = 1.0$.

6.2. Computational test of the dispatching heuristics

The proposed dispatching heuristics are tested on randomly generated problems of various sizes in terms of the number of jobs and number of machines and under various conditions of due date range and tightness.

6.2.1. Data and performance measures

The dispatching heuristic procedures described in this section were tested using the data described in section 6.1 as well as additional data generated using the same methodology (Schaller and Valente, 2019) as was used to generate the data described in section 6.1. Since we wanted to see how the dispatching heuristics performed for larger sized problems, we added additional levels of number of jobs. In addition to the four levels tested in section 4.3 ($n = 8$, 10, 12 and 15) we added an additional 7 levels: $n = 20$, 25, 30, 40, 50, 75 and 100 for the 3 levels of m (5, 10 and 20). The same nine sets of distributions of due date range and tightness tested in section 6.1 are tested. Each problem set consists of 10 instances.

A second set of 10 problems for each of the sets of parameters described above were created to use in preliminary tests to determine the parameter w for the LIN1, LIN2, H1, and H2 procedures. This parameter is multiplied by the estimated makespan to create the slack threshold (*slk_thr*). Based on these experiments a value of $w = 0.6$ was selected for LIN1, $w = 0.5$ for LIN2, $w = 0.9$ for H1, and $w = 0.5$ for h2. Three values need to be selected for parameters used in the FV procedure. The values of the parameters in the FV procedure that were selected are a = 0.90, b = 0.75, and c = 300.

Turbo Pascal was used to implement the procedures and a Dell Inspiron 1525 GHz Lap Top computer was used to conduct the test. We used two measures of performance to evaluate the procedures. For the three levels of numbers of jobs we were able to obtain optimal solutions, $n = 8$, 10 and 12, we compared the objective values obtained using dispatching procedures to the optimal objective value. The measure of performance for the small sized instances is percentage deviation (*% Dev*) of the objective value of the solution generated by each procedure from the optimal objective value. *% Dev* = $[(Z_h - Z_O)/Z_O] * 100$, where Z_O = the optimal objective value, and Z_h = the objective value of the solution generated by the dispatching heuristic procedure (EDD, MDD,

Table 3
Average % Dev versus the optimal objective value.

Procedure	Number of Machines (m)								
	5			10			20		
	Number of Jobs (n)			Number of Jobs (n)			Number of Jobs (n)		
	8	10	12	8	10	12	8	10	12
EDD	45.83	58.04	62.03	55.17	61.46	67.66	52.13	63.64	68.78
MDD	30.63	36.44	36.88	23.23	26.13	34.05	18.93	23.98	28.61
SLK	69.37	76.55	80.89	79.49	88.56	96.95	73.07	86.49	97.00
SLKP	59.10	63.49	65.06	73.96	77.63	83.89	67.54	80.80	84.84
LIN1	28.14	32.96	36.15	32.19	34.73	38.32	25.06	28.34	32.89
LIN2	34.46	40.83	39.25	37.93	42.88	45.16	29.76	32.73	38.32
CRSA	87.77	110.14	127.25	82.08	92.20	99.95	67.48	82.13	88.89
CRSB	55.12	55.17	54.47	50.02	50.91	50.00	43.49	49.49	46.44
SLRA	104.63	132.45	153.66	90.42	104.35	108.59	71.42	82.18	94.22
SLRB	80.55	91.69	94.98	60.39	67.66	69.61	51.27	55.49	52.25
FV	45.55	53.33	52.72	46.84	47.17	52.21	40.02	42.17	46.11
H1	19.53	22.72	23.14	23.61	26.10	27.80	24.50	26.68	26.55
H2	38.74	42.80	40.83	43.58	40.47	41.61	32.69	35.58	36.75

Table 4
% deviation from the EDD solution for $m = 5$.

Procedure	Number of Jobs (n)								
	15	20	25	30	40	50	75	100	Ave.
MDD	−13.72	−16.17	−18.21	−18.30	−19.93	−22.08	−23.03	−23.28	−19.34
SLK	14.08	14.48	15.55	15.19	13.33	14.00	13.33	13.70	14.21
SLKP	6.17	1.16	0.76	−2.28	−6.35	−7.25	−10.84	−11.74	−3.80
LIN1	−13.96	−16.94	−19.38	−19.66	−22.91	−24.32	−27.18	−27.61	−21.50
LIN2	−8.61	−13.48	−15.74	−16.41	−19.93	−20.32	−22.25	−22.32	−17.38
FV	−1.42	−4.44	−4.86	−4.38	−9.12	−9.06	−10.82	−11.27	−6.92
CRSA	33.65	36.99	38.16	30.74	27.34	26.45	23.31	19.61	29.53
CRSB	−2.64	−7.61	−9.21	−10.41	−15.97	−18.40	−20.93	−21.09	−13.28
SLRA	49.43	49.72	49.34	42.78	37.73	35.54	30.32	25.94	40.10
SLRB	18.11	13.62	6.91	3.00	−6.45	−10.16	−15.33	−17.35	−0.96
H1	−20.03	−20.20	−23.70	−22.58	−26.79	−27.12	−29.70	−29.95	−25.01
H2	−6.64	−10.61	−10.51	−13.21	−16.80	−16.63	−19.24	−20.34	−14.25

LIN1, LIN2, SLK, SLKP, FV, H1, H2). The measure of performance used to evaluate the dispatching procedures for the larger sized instances ($n \geq 15$) is percentage deviation (*% Dev*) of the objective value of the solution generated by each procedure from the objective value of the solution generated by the EDD procedure. *% Dev* = $[(Z_h - Z_{EDD})/Z_{EDD}] * 100$, where Z_{EDD} = the objective value of the solution generated by the EDD procedure, and Z_h = the objective value of the solution generated by the dispatching heuristic procedure (MDD, LIN1, LIN2, SLK, SLKP, FV, CRSA, CRSB, SLRA, SLRB, H1, H2). The EDD procedure would always have a *% Dev* equal to 0.00 so it is omitted. The results of these tests are presented in the next sections.

6.2.2. Results of the test versus the optimal objective value

Table 3 shows the average *% Dev* for each procedure for each level of number machines and jobs.

The results show that the H1 and MDD procedures were the two best performing procedures. The H1 procedure was best on six of the nine combinations of n and m and was second on the other three. The MDD procedure was best for three of the nine combinations, was second for three combinations, and was third for three combinations. The LIN1 procedure was generally the third best performing procedure having the third best performance for six of the nine combinations and was second for three of the combinations. The LIN2 and H2 procedures performance followed the above three procedures with LIN2 better than H2 for seven of the nine combinations. The SLRA and SLK procedures performed the

Table 5
% deviation from the EDD solution for $m = 10$.

Procedure	Number of Jobs (n)								
	15	20	25	30	40	50	75	100	Ave.
MDD	−21.67	−20.83	−24.37	−25.63	−26.83	−27.67	−29.77	−30.95	−25.97
SLK	18.25	18.23	18.55	19.64	17.91	19.53	18.39	17.93	18.55
SLKP	8.35	5.26	2.51	0.00	−3.09	−4.96	−10.96	−13.36	−2.03
LIN1	−20.12	−19.00	−22.62	−22.90	−25.17	−26.66	−30.05	−31.79	−24.79
LIN2	−15.83	−16.34	−19.84	−19.93	−21.93	−22.76	−26.36	27.33	−21.29
FV	−7.34	−10.33	−12.23	−14.03	−17.07	−17.53	−20.15	−22.24	−15.12
CRSA	26.24	23.34	19.87	20.43	20.28	22.03	18.16	16.85	20.90
CRSB	−7.96	−13.92	−19.13	−20.72	−24.78	−26.56	−31.85	−34.63	−22.44
SLRA	32.32	29.63	25.26	26.17	26.87	27.27	23.18	21.01	26.46
SLRB	2.63	−1.37	−8.85	−11.16	−15.91	−18.64	−25.75	−29.92	−13.62
H1	−24.09	−25.14	−26.71	−27.35	−31.35	−32.10	−34.32	−36.54	−29.70
H2	−13.13	−13.84	−17.37	−17.48	−20.38	−20.69	−22.98	−24.72	−18.82

Table 6
% deviation from the EDD solution for $m = 20$.

Procedure	Number of Jobs (n)								
	15	20	25	30	40	50	75	100	Ave.
MDD	−26.17	−26.18	−28.44	−28.79	−29.20	−30.81	−32.51	−33.47	−29.45
SLK	17.17	15.88	16.05	19.31	18.70	19.48	19.33	19.79	18.21
SLKP	7.91	4.55	2.49	2.22	−2.38	−5.59	−10.31	−12.83	−1.74
LIN1	−23.48	−22.83	−26.47	−26.22	−27.37	−28.80	−31.19	−32.69	−27.38
LIN2	−20.33	−20.82	−23.12	−23.81	−25.65	−27.26	−28.78	−30.28	−25.01
FV	−15.90	−16.29	−21.29	−20.77	−23.02	−27.26	−26.53	−28.57	−22.45
CRSA	6.99	14.48	7.50	11.30	13.02	10.83	12.47	11.97	11.07
CRSB	−15.69	−18.90	−25.88	−26.84	−28.32	−33.00	−35.62	−37.79	−27.76
SLRA	8.61	15.70	8.96	13.85	15.58	12.46	14.01	13.95	12.89
SLRB	−12.41	−15.32	−22.10	−21.79	−23.37	−29.06	−31.30	−34.38	−23.72
H1	−25.91	−25.96	−30.62	−31.85	−33.06	−35.23	−37.02	−38.93	−32.32
H2	−20.64	−20.50	−24.09	−24.40	−25.53	−26.26	−27.93	−29.72	−24.88

Table 7
% deviation from the EDD solution by r for $n = 50$ and $m = 10$.

r	Procedure											
	MDD	LIN1	LIN2	SLK	SLKP	FV	CRSA	CRSB	SLRA	SLRB	H1	H2
0.0	−22	−21	−17	20	−0.2	−14	31	−21	39	−11	−26	−14
0.2	−29	−27	−23	21	−4.8	−17	24	−28	30	−19	−34	−20
0.4	−32	−32	−26	17	−9.9	−22	11	−31	13	−26	−36	−28

worst for all of the combinations. The SLKP, CRSA, CRSB and SLRB procedures also performed poorly for all the combinations. For the CRS and SLR rules the versions that included the offset time (CRSB and SLRB) performed better than their counter parts (CRSA and SLRA).

The results also show that there is a considerable gap between the objective values for the solutions generated by the dispatching heuristics and the optimal objective value. The H1 procedure was the only procedure with an average *% Dev* under 30% for each combination and even this procedure had an average *% Dev* greater than 19% for each combination. The MDD procedure had an average *% Dev* that was less than 40% for each combination but none of the other procedures was able to average less than 40% for each combination. The SLK, SLKP, CRSA, SRLA and SLRB procedures had an average *% Dev* that were greater than 50% for every combination and the EDD and FV procedures had an average *% Dev* that was greater than 40% for every combination.

6.2.3. Results of the test for the larger sized problems versus the EDD objective value

Tables 4–6 show the *% Dev* for larger sized problems ($n \geq 15$) 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 3 shows the results for $m = 5$, Table 4, for $m = 10$ and Table 5, for $m = 20$. The results in these tables are compared with the objective value that was generated by the EDD procedure, therefore the EDD procedure will always have a *% Dev* equal to 0.00 and has been omitted.

The results show that the H1 procedure had the lowest average *% Dev* for each of the three levels of m (5, 10 and 20). This procedure was also very consistent and ranked first for each level of numbers of jobs when $m = 5$ and 10 and ranked first for each level of numbers of jobs when $m = 20$ and $n \geq 25$. Therefore, the H1 procedure ranked first in terms of *% Dev* for 22 of the 24 combinations of n and m. For the two combinations the H1 procedure was not the best, $m = 20$ and $n = 15$ and 20, it was the second best procedure (MDD was the best performing procedure for these two combinations). The H1 procedure was also the only procedure that had an average *% Dev* that less than −20% for each combination of n and m.

For $m = 5$ and 10 the next two best performing procedures were the MDD and LIN1 procedures. These procedures were ranked either second or third on all the combinations of n when $m = 5$. When $m = 10$, MDD was the second best and LIN1, third best for $n < 50$, as n increased the performance of CRSB improved and this procedure was third best for $n = 50$, and second best for $n = 75$ and 100. When $m = 20$, MDD was second best for $n < 50$ and CRSB was second best for $n \geq 50$.

As with the small sized problems, the SLK, SLKP, CRSA, SLRA procedures performed the poorest. For each combination of n and m the SLK, CRSA and SLRA procedures had a positive *% Dev* and SLKP procedure had a positive *% Dev* for $n < 30$ for all three levels of m. Also, as in the small sized problems, for the CRS and SLR rules the versions that included the offset time (CRSB and SLRB) performed better than their counter parts (CRSA and SLRA).

In order to show the effect of the due date range (R) and tardiness factor (r) on the results Tables 7 and 8 are presented. Table 7 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 H1 procedure performed the best for each level of r for this setting of n and m. For each level of r, The MDD procedure ranked second. Each of the procedures with the exception of the SLK procedure performed better relative to the EDD procedure as r increased.

Table 8 shows the *% Dev* by due date range factor (R) for $n = 50$ and $m = 10$.

As in the results by r, the results by due date range factor (R) show that the H1 procedure was consistent across the levels of R for this

Table 8
% deviation from the EDD solution by R for $n = 50$ and $m = 10$.

R	Procedure											
	MDD	LIN1	LIN2	SLK	SLKP	FV	CRSA	CRSB	SLRA	SLRB	H1	H2
0.2	−24	−22	−20	24	−9.0	−14	7	−29	8.0	−28	−28	−14
0.6	−29	−27	−23	15	−4.1	−19	20	−27	26	−20	−33	−22
1.0	−31	−31	−26	19	−1.7	−20	39	−24	48	−8.4	−36	−26

setting of n and m, and was best or second best for each level. The CRSB procedure was best when $R = 0.2$. The MDD, LIN1, LIN2, FV, H1 and H2 procedures all performed better relative to the EDD procedure as R increased. The SLKP, CRSA, CRSB, SLRA and SLRB procedures all performed worse relative to the EDD procedure as R increased.

The procedures are efficient and able to solve the instances quickly. Computational times do increase as problem sizes become large, but the procedures averaged less than 0.5 s per instance for all problem sizes.

Since the H1 procedure generally generated the best solutions and was very consistent it is the recommended dispatching procedure.

7. Improvement procedure

In section 6.2.2 the objective values of the solutions generated by the dispatching procedures were compared to optimal objective values for small sized problems. The results of this test showed that there was a large gap between the objective values generated by the dispatching heuristics and the optimal values. Therefore, we propose an improvement procedure in this section to develop better solutions.

7.1. Insertion neighborhood search procedure

Our proposed procedure searches a neighborhood by removing a job and inserting it into a different position in the sequence. In this procedure we start with a sequence σ and convert it to a different sequence σ' by changing the position of one of the jobs. To search the neighborhood, we remove each job from the sequence σ and create $n - 1$ trial sequences by inserting the removed job into each possible position while the order of the other jobs is maintained. If a solution that is better than the incumbent sequence is obtained, then the incumbent is updated. The insertion search is repeated until it fails to find an improvement and is terminated.

The procedure needs an initial solution to start. In our implementation we use the H1 procedure to generate the initial solution. This procedure was chosen because it was the best dispatching procedure among the procedures tested. This procedure is referred to as the H1INS procedure in this paper.

To evaluate a neighborhood using the insertion procedure O (n^2) insertions are needed. Since each sequence created by an insertion can be evaluated in O (n) time the complexity of the procedure is O (n^3) each time a neighborhood is evaluated. As the number of jobs increases this procedure can be time consuming but speed up methods have been developed for using this search in the no-wait flow shop environment for other objectives. In the next section we present a speed up method for this objective.

7.2. Speed up procedure

Approaches to improve the efficiency of insert operations for improvement procedures have been used for the no-wait flow shop environment. Li et al. (2008) developed a property that reduced the evaluation of the makespan from O (n) to O (1) for a partial sequence when a new job is inserted. This property was used by Li et al. (2008) and Pan et al. (2008) to develop heuristics with greater efficiency.

Our objective is different than the makespan objective because a job's earliness or tardiness is a nonlinear function of the job's completion time. Ding et al. (2015) developed a speed up procedure for insertion operations for a no-wait flow shop with minimizing total tardiness as the objective, which is similar to our objective because a job's tardiness is a nonlinear function of its completion time. Ding et al. (2015) use the lateness of the jobs, which is a linear function of completion time to develop properties that allow for greater efficiency in evaluating insertion operations. They break the set of jobs into sensitive jobs and non-sensitive jobs. For example, if there are tardy or on time jobs in a partial sequence and a new job is inserted before these jobs then these jobs will all increase in tardiness by the amount of displacement

Table 9
Average *% Dev* from the optimal solution for the H1INS procedure.

Number of Jobs (n)	Procedure					
	H1INS			H1		
	Number of Machines (m)			Number of Machines (m)		
	5	10	20	5	10	20
8	2.22	2.68	2.18	19.53	23.61	24.50
10	3.49	3.86	2.60	22.72	26.10	26.68
12	2.84	4.58	3.76	23.14	27.80	26.55

caused by the insertion of the new job, so these jobs are non-sensitive jobs. Therefore, the increase in tardiness for the set of tardy or on time jobs can be calculated in O (1) time. Jobs that are sufficiently early so they will not become tardy if a new job is inserted before them can also be identified and these jobs will be non-sensitive because they will have no increase in tardiness. The set of jobs that are early before an insertion and become tardy after the insertion are sensitive jobs and need to have the change in the objective calculated individually. Ding et al. (2015) developed heuristics with insertion operations and in computational tests found that the set of sensitive jobs was usually small relative to the set of non-sensitive jobs, so the speed procedure resulted in the heuristics having greater efficiency. With our objective we also have to consider the effect of an insertion on early jobs, but this does not seem to be too difficult and the basic elements of Ding et al. (2015)'s speed up procedure can be used with some modification for the early/tardy no-wait flow shop problem.

To search a neighborhood, we remove each job from its current position in a current sequence and insert it into each position of the sequence maintaining the order of the other jobs. Let j_c be the job that is removed from the current sequence. When job j_c is removed we have a sequence of $n - 1$ jobs. Let σ_0 represent this sequence. Let $Z(\sigma_0)$ equal the total earliness and tardiness of the jobs in the sequence σ_0. We create n sequences ($\sigma_1, \sigma_2, \ldots, \sigma_n$) by inserting job j_c before each position k ($k = 1, \ldots, n - 1$) in σ_0 and at the end of σ_0.

Initially, before the insertions, we perform three steps. In the first step we calculate the total earliness and tardiness of the jobs in σ_0, as well as the earliness or tardiness of each job. As we insert job j_c into position k of the sequence the jobs that were in positions k through $n - 1$ are all shifted later by a certain amount of time. Let RS_k equal the amount of time that the jobs in positions k through $n - 1$ will have their completion time increase when job j_c is inserted into position k. Let $j_{[k-1]}$ be the job before the job in position k ($j_{[k-1]} = 0$ if $k = 1$) in σ_0 and $j_{[k]}$ be the job in position k in σ_0. Then $RS_k = OS_{jc,j[k-1]} + p_{jc,1} + OS_{j[k],jc} - OS_{[k],[k-1]}$. Let $MaxRS = \max\{RS_k\}$ for $k = 1, \ldots, n - 1$. This is the second step. In the third step we divide the jobs in σ_0 into three sets and create indexes for the jobs that fall into each of the sets. The three sets are the set of jobs which are on-time or tardy τ_T, the set of jobs with earliness greater than or equal to $MaxRS$ ($E_k \geq MaxRS$) τ_E, and the set of sensitive jobs τ_S ($0 \leq E_k < MaxRS$). As we insert job j_c into each position to create a new sequence with n jobs we start with position 1 and work to position n. Each time we create a new sequence we update the three sets by taking the job in the prior position out of its appropriate set. Let *ntdy* be the number of jobs in the set τ_T, and *nely* be the number of jobs in the set τ_E. Let $\Delta Z(\tau_T)$ be the change in the total tardiness of the jobs in the set τ_T caused by the insertion of job j_c, $\Delta Z(\tau_E)$ be the change in the total earliness of the jobs in the set τ_E caused by the insertion of job j_c, and $\Delta Z(\tau_S)$ be the change in the total earliness and tardiness of the jobs in the set of sensitive jobs caused by the insertion job job j_c. Let $Z(\sigma_k)$ equal the total earliness and tardiness of the sequence σ_k formed by inserting job j_c before position k of the sequence σ_0. $Z(\sigma_k) = Z(\sigma_0) + \Delta Z(\tau_T) + \Delta Z(\tau_E) + \Delta Z(\tau_S)$. $\Delta Z(\tau_T) = ntdy^* RS_k$ and $\Delta Z(\tau_E) = -nely^* RS_k$. Therefore, the change in the objective for the jobs in the two sets τ_T and τ_E caused by the insertion of job j_c into position k can be calculated in O (1) time. We need to calculate the change in total earliness and tardiness for each of the

Table 10
Average *% Dev* from the EDD solution for the H1INS procedure.

Number of Jobs (n)	Procedure					
	H1INS			H1		
	Number of Machines (m)			Number of Machines (m)		
	5	10	20	5	10	20
15	−33.91	−40.37	−41.37	−20.03	−24.09	−25.91
20	−36.27	−41.62	−41.67	−20.20	−25.14	−25.96
25	−39.05	−44.09	−46.46	−23.70	−26.71	−30.62
30	−38.69	−44.58	−46.88	−22.58	−27.35	−31.85
40	−43.30	−47.18	−48.91	−26.79	−31.35	−33.06
50	−43.84	−48.28	−50.15	−27.12	−32.10	−35.23
75	−46.33	−50.48	−51.64	−29.70	−34.32	−37.02
100	−46.90	−52.00	−52.86	−29.95	−36.54	−38.93
AVE.	−41.04	−47.20	−47.49	−25.01	−29.70	−32.32

jobs in the set τ_S by going through the index of these jobs each time we are checking a sequence. If *nsens* is the number of sensitive jobs (set τ_S) then it will take O (*nsens*) time to calculate the change in the objective for these jobs. If *nsens* is much smaller than n then there will be a significant time saving by using this approach as opposed to checking all the jobs one by one. This procedure is referred to as the H1INB procedure.

7.3. Computational results

We tested the insertion neighborhood search procedure using the data described in subsections 6.1 and 6.2.1. The first comparison uses the objective values of the optimal solutions for the small sized instances.

7.3.1. Comparison with optimal objective values

Table 9 compares the solutions generated by the H1INS procedure with the optimal solutions that were generated for the small sized problems. The measure of performance is the percentage deviation (*% Dev*) which is calculated as: $\%Dev = [(Z_h - Z_O)/Z_O] * 100$, where $Z_O =$ the optimal total earliness and tardiness, and $Z_h =$ the total earliness and tardiness of the solutions generated by the H1INS procedure. The H1 procedure's results were also included in the table to help see the improvement achieved by using the insertion search.

The results show that using the insertion search significantly improved the solutions for the small sized problems. The average *%Dev* was less than 5% for each of the combinations of n and m for the small sized problems.

7.3.2. Comparison with EDD objective values for larger sized instances

Table 10 compares the solutions generated by the H1INS procedure with the solutions generated by the EDD for the large sized problems. The measure of performance is the percentage deviation (*%Dev*) which is calculated as: $\%Dev = [(Z_h - Z_{EDD})/Z_{EDD}] * 100$, where $Z_{EDD} =$ the objective value of the solution generated by the EDD procedure, and $Z_h =$ the objective value of the solution generated by the H1INS procedure. The H1 procedure's results were also included in the table to help see the improvement achieved by using the insertion search.

The results show that using the insertion search significantly improved the solutions for the large sized problems. This can be seen most easily by looking at the last row of the table that has the overall average *% Dev* for each level of number of machines. The averages with the insertion search included show the improvement versus the EDD solution was over 40% for each level of number of machines. Also, the improvement was quite large for each combination of n and m compared to just using the H1 procedure.

7.3.3. Comparison between the H1INS and H1INB (speedup) procedures

When the H1INS and H1INB procedures were run we recorded the amount of time in seconds that was needed to generate the solution for each instance. Both procedures were coded in Turbo Pascal and were tested on a Dell Inspiron 1525 GHz Lap Top computer. The two procedures both use an insertion search to improve upon the initial solution generated by the H1 procedure but the H1INB procedure incorporates methods to speed up the process. Table 11 shows the average time in seconds, by n and m, per instance required by each of the procedures. Also, there is a column that shows the percent reduction in seconds needed when using the H1INB procedure instead of the H1INS procedure.

The results show that as the number of jobs increases the time required by both procedures increases. The results also show that using the speed up methods results in a significant reduction in the time needed to generate solutions. In addition, it appears that as the number of jobs increases the amount of benefit obtained by using the speed up methods increases. For example, when $n = 100$ the greatest percentage reduction occurred for all three levels of m and averaged 90%. These results show that using the insertion search improvement procedure with the speed up methods allows relatively good solutions to be obtained in a reasonable amount of time for larger size problems.

8. Conclusion

This paper considers the problem of minimizing total earliness and tardiness in a no-wait flow shop. An exact branch-and-bound algorithm is proposed for the problem. This branch-and-bound algorithm was test on small sized problems and was found to be able to solve very small sized problems but the increase in computational time quickly becomes prohibitive as the number of jobs increases. Therefore, heuristic procedures were investigated. Twelve dispatching heuristics were tested for the problem. Ten of these heuristics were based on heuristics used in other environments with some minor modifications. The two heuristics,

Table 11
CPU seconds by n and m for the insertion search procedures.

n	Number of Machines (m)								
	5			10			20		
	H1INS	H1INB	% Red	H1INS	H1INB	% Red	H1INS	H1INB	% Red
8	0.010	0.008	20	0.031	0.009	71	0.053	0.020	62
10	0.049	0.023	53	0.053	0.026	51	0.054	0.029	46
12	0.034	0.013	62	0.030	0.009	70	0.057	0.032	44
15	0.051	0.034	33	0.050	0.027	46	0.057	0.036	37
20	0.086	0.041	52	0.097	0.044	55	0.099	0.046	54
25	0.176	0.077	56	0.191	0.079	59	0.190	0.073	62
30	0.320	0.114	64	0.352	0.107	70	0.334	0.107	68
40	1.009	0.257	75	0.988	0.228	77	1.009	0.226	78
50	2.413	0.496	79	2.53	0.480	81	2.354	0.416	82
75	12.13	1.817	85	13.33	1.736	87	12.52	1.502	88
100	40.68	4.58	89	42.13	4.26	90	40.04	3.62	91

that were newly proposed, utilize some of the characteristics specific to the no-wait flow shop in order to sequence the jobs. One of these heuristics, referred to as the H1 procedure, was generally found to be the best performing heuristic.

When the dispatching heuristics were tested on small sized problems and the solutions were compared with the optimal solutions generated by the branch-and-bound procedure it was found a significant gap existed in the quality of the solutions for even the best performing dispatching heuristics (H1 and MDD). Therefore, an insertion search improvement procedure was proposed for the problem. Two insertion search improvement procedures were developed. One included speed up procedures that use the structure of the problem. It was found that including the insertion search improvement procedure resulted in much improved solutions. Also using the speed up methods, enable the procedure to generate solutions for larger sized problems in a reasonable amount of time.

A future area of additional research would be the development of metaheuristics that generate better solutions for the problem. Incorporating the dispatching heuristics and the insertion search with the speed up methods could be a foundation for some of the metaheuristic approaches such as genetic algorithm or tabu search.

Declaration of competing interest

None.

References

Aldowaisan, T., Allahverdi, A., 2012. Minimizing total tardiness in no-wait flowshops. Found. Comput. Decis. Sci. 37, 149–162.

Anderson, E.J., Nyirenda, J.C., 1990. Two new rules to minimize tardiness in a job shop. Int. J. Prod. Res. 28 (12), 2277–2292.

Baker, K.R., Bertrand, J.W.M., 1982. A dynamic priority rule for scheduling against due-dates. J. Oper. Manag. 3 (1), 37–42.

Baker, K.R., Scudder, G.D., 1990. Sequencing with earliness and tardiness penalties: a review. Oper. Res. 38, 22–36.

Chandra, C., Mehta, P., Tirupati, D., 2009. Permutation flow shop scheduling with earliness and tardiness penalties. Int. J. Prod. Res. 47, 5591–5610.

Davis, J.S., Kanet, J.J., 1993. Single-machine scheduling with early and tardy completion costs. Nav. Res. Logist. 40, 85–101.

Ding, J., Song, S., Zhang, R., Gupta, J.N.D., Wu, C., 2015. Accelerated methods for total tardiness minimization in no-wait flowshops. Int. J. Prod. Res. 53 (4), 1002–1018.

Fernandez-Viagas, V., Dios, M., Framinan, J.M., 2016. Efficient constructive and composit heuristics for the permutation flowshop to minimize total earliness and tardiness. Comput. Oper. Res. 70, 38–68.

Framinan, J.M., Nagano, M.S., Moccellin, J.V., 2010. An efficient heuristic for total flowtime minimization in no-wait flowshops. Int. J. Adv. Manuf. Technol. 46 (9–12), 1049–1057.

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

Gao, K., Pan, Q., Li, J., 2011. Discrete harmony search algorithm for the no-wait flow shop scheduling problem with total flow time criterion. Int. J. Adv. Manuf. Technol. 56 (5–8), 683–692.

Hall, N.G., Sriskandarajah, C., 1996. A survey of machine scheduling problems with blocking and no-wait in process. Oper. Res. 44 (3), 510–525.

Hoogeveen, H., 2005. Multicriteria scheduling. Eur. J. Oper. Res. 167, 592–623.

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

Kanet, J.J., Sridharan, V., 2000. Scheduling with inserted idle time: problem taxonomy and literature review. Oper. Res. 48, 99–110.

Kim, Y.D., Yano, C.A., 1994. Minimizing mean tardiness and earliness in single-machine scheduling problems with unequal due dates. Nav. Res. Logist. 41, 913–933.

Korman, K., 1994. A Pressing Matter, pp. 46–50. Video.

Laha, D., Chakraborty, U.K., 2009. A constructive heuristic for minimizing makespan in no-wait flow shop scheduling. Int. J. Adv. Manuf. Technol. 41 (1–2), 97–109.

Landis, K., 1993. Group Technology and Cellular Manufacturing in the Westvaco Los Angeles VH Department. School of Business, University of Southern California. Project report in IOM 581.

Li, X., Wang, Q., Wu, C., 2008. Heuristic for no-wait flow shops with makespan minimization. Int. J. Prod. Res. 46 (9), 2519–2530.

Liu, G., Song, S., Cheng, W., 2013. Some heuristics for no-wait flowshops with total tardiness criterion. Comput. Oper. Res. 40 (2), 521–525.

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. J. Oper. Res. Soc. 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. Int. J. Prod. Econ. 122, 763–773.

M'Hallah, R., 2014. An iterated local search variable neighborhood descent hybrid heuristic for the total earliness tardiness permutation flow shop. Int. J. Prod. Res. 52 (13), 3802–3819.

Ow, P.S., Morton, T.E., 1989. The single-machine early tardy problem. Manag. Sci. 35 (2), 177–191.

Pan, Q., Wang, L., Zhao, B., 2008. An improved iterated greedy algorithm for the no-wait flow shop scheduling problem with makespan criterion. Int. J. Adv. Manuf. Technol. 38 (7–8), 778–786.

Panwalkar, S.S., Iskander, W., 1977. Survey of scheduling rules. Oper. Res. 25 (1), 45–61.

Raghu, T.S., Rajendran, C., 1993. An efficient dynamic dispatching rule for scheduling in a job shop. Int. J. Prod. Econ. 32 (3), 301–313.

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. Int. J. Prod. Res. 37, 1137–1158.

Schaller, J.E., 2007. A comparison of lower bounds for the single-machine early tardy problem. Comput. Oper. Res. 34, 2279–2292.

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. J. Oper. Res. Soc. 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. Int. J. Prod. Res. 51 (3), 772.

Schaller, J.E., Valente, J.M., 2019. Heuristics for scheduling jobs in a permutation flow shop to minimize total earliness with unforced idle time allowed. Expert Syst. Appl. 119, 76–386, 119.

Szwarc, W., 1993. Adjacent ordering in single-machine scheduling with earliness and tardiness penalties. Nav. Res. Logist. 40 (2), 229–244.

Taillard, E., 1993. Benchmarks for basic scheduling problems. Eur. J. Oper. Res. 64, 278–285.

Valente, J.M.S., 2009. Beam search heuristics for the single machine scheduling problem with linear earliness and quadratic tardiness costs. Asia Pac. J. Oper. Res. 26, 319–339.

Vepsalainen, A.P.J., Morton, T.E., 1987. Priority rules for job shops with weighted tardiness costs. Manag. Sci. 33 (8), 1035–1047.

Vinod, V., Sridharan, R., 2011. Simulation modeling and analysis of due-date assignment methods and scheduling decision rules in a dynamic job shop production system. Int. J. Prod. Econ. 129 (1), 127–146.

Xiong, H., Fan, H., Jiang, G., Li, G., 2017. A simulation-based study of dispatching rules in a dynamic job shop scheduling problem with batch release and extended technical precedence constraints. Eur. J. Oper. Res. 257 (1), 13–24.

Yano, C.A., Kim, Y.-D., 1991. Algorithms for a class of single machine weighted tardiness and earliness problems. Eur. J. Oper. Res. 52, 161–178.

Zegordi, S.H., Itoh, K., Enkawa, T., 1995. A knowledgeable simulated annealing scheme for the early/tardy flow shop scheduling problem. Int. J. Prod. Res. 33, 1449–1466.