Computers & Industrial Engineering 119 (2018) 146–156

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

Efficient heuristics for minimizing weighted sum of squared tardiness on identical parallel machines

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

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

A R T I C L E I N F O

Keywords:
Scheduling
Sequencing
Heuristics
Integer programming

A B S T R A C T

Scheduling jobs on a set of identical parallel machines using efficient heuristics when the objective is to minimize total weighted squared tardiness is considered. Two efficient heuristics and an improvement procedure are presented for the problem. These heuristics and other heuristics are tested using problem sets that represent a variety of conditions. The results show that one of the heuristics consistently performs better than the other heuristics tested. It is also shown how these heuristics can be incorporated into other procedures such as the existing Lagrangian relaxation procedure or meta-heuristics to obtain improved solutions for medium sized problems.

1. Introduction and problem description

This research investigates the use of efficient heuristics when scheduling on identical parallel machines with an objective of minimizing total weighted squared tardiness. Let M be the number of machines and n the number of jobs to be scheduled. Let p_j, w_j, and d_j represent the processing time, weight and due date for each job j (j = 1, …, n) respectively, and C_j is the time that job j is completed, j = 1, …, n. The weights (w_j) for each job would usually be determined by the scheduler or other management individuals in the organization and would reflect things such as product margins or importance of the customer to the organization's success. The tardiness of job j, T_j, is defined as: $T_j = \max\{0, C_j - d_j\}$, j = 1, …, n. The objective function, Z, can be expressed as: $Z = \sum_{j=1}^{n} w_j * T_j^2$. If the job to be sequenced in position j of machine m (m = 1, …, M) is denoted as [j]m, then $C_{[j]m} = C_{[j-1]m} + p_{[j]}$, with $C_{[0]m} = 0$ for m = 1, …, M.

The problem described in the previous paragraph was originally motivated by studies of scheduling in an Aerospace and Defense manufacturing facility (Hoitomt, Luh, Max, & Pattipati, 1990; Luh & Hoitomt, 1993). A distinguishing feature of the problem is that the objective is a function of the square of tardiness. Manufacturers, as well as service organizations, operate as part of supply chains in which timely delivery to customers is crucial, and the cost of tardy deliveries can be very high. Taguchi (1986) proposed a quadratic penalty to measure quality costs incurred by customers. Timely delivery to meet requested customer due dates is a dimension of quality that is very important. Quadratic tardiness can be used as part of a measure of on time delivery to represent the increased cost as tardiness increases, as suggested by Taguchi (1986). Traditionally, linear functions of tardiness have been used to evaluate schedules. The sum of weighted tardiness would be the linear equivalent of the objective considered in this paper. Frequently, the solution can be different depending on whether weighted tardiness or weighted squared tardiness is used as the objective. For example suppose two jobs are to be sequenced first and second on the same machine and have these processing times, weights and due dates: $p_1 = 6$, $p_2 = 3$, $d_1 = 4$, $d_2 = 0$, $w_1 = 3$, $w_1 = 1$. If weighted tardiness is used as the objective then scheduling job 1 before job 2 would result in a total weighted tardiness of 15 for the two jobs. If job 2 is instead scheduled before job 1, the total weighted tardiness of the two jobs would be 18, so scheduling job 1 before job 2 would be better. If weighted squared tardiness is used as the objective then scheduling job 1 before job 2 would result in a total weighted squared tardiness of 93 for the two jobs. If job 2 is instead scheduled before job 1, the total weighted squared tardiness of the two jobs would be 84, so scheduling job 2 before job 1 would be better and therefore the solutions under the two objectives would be different. All customers are considered with either of these objectives. Another objective, maximum tardiness, has also been frequently used. This is a simple measure and implicitly recognizes that customer dissatisfaction with tardiness does not increase in a linear fashion, as a schedule with one job that is 10 units tardy is worse than a schedule that has two tardy jobs, each five units tardy. In fact, if squared tardiness was used, the first schedule in the preceding example would be twice as costly as the second (100 versus 50). A problem with using maximum tardiness is that it focuses on just the one

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

https://doi.org/10.1016/j.cie.2018.03.036
Received 29 August 2017; Received in revised form 20 March 2018; Accepted 21 March 2018
Available online 26 March 2018
0360-8352/ © 2018 Elsevier Ltd. All rights reserved.

job or customer that has the maximum tardiness, whereas squared tardiness considers all of the jobs or customers. Sun, Noble, and Klein (1999) provide additional examples contrasting squared tardiness with linear tardiness and maximum tardiness. In situations where demand is higher than the capacity of the machines or processors, the costs of tardy deliveries can become very high. Sometimes it is possible to increase capacity by using subcontractors, so the total weighted squared tardiness can be reduced. It is important to be able to generate timely schedules that would indicate the problem and allowing for searching for alternatives for meeting customer demand. If subcontractors are not available, then these schedules would allow for the evaluation of adding additional machines or processors.

To the best of the authors' knowledge, the complexity of the weighted squared tardiness on parallel machines is open. Other complexity results suggest that this problem is likely hard. Lawler (1977) and Lenstra, Rinnooy Kan, and Brucker (1977), show that the single machine weighted tardiness problem is strongly NP-hard. Garey and Johnson (1978), show that minimizing makespan on parallel machines is strongly NP-hard. Schaller and Valente (2013), developed exact algorithms for the problem, but were only able to solve small scale sized problems with up to 15 jobs and four machines in a reasonable amount of time. For these reasons we focus on efficient heuristics that can solve larger sized problems. Our objective is to see if efficient heuristics that have been used for the single-machine problem with the same objective can be adapted to the parallel machines environment, in order to generate good solutions for the problem quickly. The next section will review literature for problems with the objective of minimizing weighted squared tardiness. Efficient heuristics are presented for the problem in section three, and section four describes an improvement procedure that can be applied to solutions generated by the heuristics. Computational tests are described and results presented in section five. It is shown how the procedures can be incorporated into other procedures, in order to improve performance, in section six. Finally, the last section concludes the paper.

2. Literature review

Past research that is related to this paper includes research with a tardiness objective and the parallel machines environment. Sen, Sulek, and Dileepan (2003), provide a survey of research for minimizing weighted and unweighted tardiness. Mokotoff (2001), surveyed research for parallel machine scheduling problems. The focus of this literature review will be research with the objective of minimizing the weighted sum of squared tardiness. Previous work on problems involving an objective of minimizing the sum of weighted squared tardiness includes single-machine problems (Goncalves et al., 2016; Schaller & Valente, 2012; Sun et al., 1999; Valente & Schaller, 2012) parallel machine problems (Hoitomt et al., 1990; Luh & Hoitomt, 1993; Schaller & Valente, 2013) and job shop problems (Sun & Noble, 1999; Thomalla, 2001). Several papers, Hoitomt et al., 1990; Luh & Hoitomt, 1993; Sun et al., 1999; Sun & Noble, 1999; Thomalla, 2001, use a Lagrangian relaxation approach based on the procedure that Fisher (1973a, 1973b) used for other problems.

Hoitomt et al. (1990), studied a parallel machines shop environment at a Pratt and Whitney plant where the jobs have precedence constraints. A Lagrangian relaxation procedure was developed and demonstrated using three examples from the plant. Luh and Hoitomt (1993), also developed a Lagrangian relaxation procedure for a shop environment at a Pratt and Whitney plant that consisted of identical parallel machines. This procedure was also tested using data from the plant.

Sun et al. (1999), studied a single machine problem that included sequence-dependent setup times and release dates. They also developed a lagrangian relaxation procedure and compared their procedure against simple dispatching rules as well as more computationally intensive algorithms, namely tabu search and simulated annealing.

A job shop environment was studied by Sun and Noble (1999) and Thomalla (2001). In both papers lagrangian relaxation procedures were developed. The problem Sun and Noble (1999), studied included sequence-dependent setup times and Thomalla (2001)'s, included alternative processing plans.

Schaller and Valente (2012) and Goncalves et al. (2016), all considered the single machine problem. Schaller and Valente (2012), proposed several dominance rules, as well as branch-and-bound algorithms, which incorporate these rules, and found that problems with up to 50 jobs could be solved within a reasonable amount of time. Valente and Schaller (2012), focused on developing dispatching heuristics for the problem in order to solve large scale instances. They found that a rule that sequences jobs from the end of the schedule and works toward the beginning of the schedule worked the best. Goncalves et al. (2016), developed meta-heuristics for the problem that included an iterated local search, a variable greedy algorithm and a genetic algorithm. In computational tests the iterated local search generated solutions with lower objective values.

Schaller and Valente (2013) developed branch-and-bound algorithms for the identical parallel machines problem. These branch-and-bound algorithms utilized results from the single machine scheduling problem, but it was found that only small scale problems of up to 15 jobs could be solved in a reasonable amount of time.

3. Heuristics

In this section two heuristics are described and proposed for the problem. These two procedures are referred to as the QB and QBP procedures. Both of these procedures are adaptations to parallel machines of the single machine heuristic QBackv6 developed by Valente and Schaller (2012). The QBackv6 heuristic was found by Valente and Schaller (2012) to be the best performing heuristic for the single machine problem. The QBackv6 heuristic works from the end of a sequence to the beginning. If n is the number of jobs to be sequenced then, at each iteration k of the procedure, the jobs in positions n–k to n are sequenced and the jobs not yet sequenced need to be assigned to the first n–k–1 positions. This allows for the jobs that will be at the end of the sequence, and completed the latest, to be examined first and hopefully avoid placing jobs with a high cost in these positions. In the single machine problem this approach is straightforward because we know a job's completion time when it is sequenced, even though we do not know the exact order of the jobs to be sequenced before it on the machine. In the parallel machines case this is not so; therefore, the two approaches described in this section attempt to estimate a job's completion time when determining the sequence. The two procedures differ in the way that they estimate completion times. Once a sequence is determined, each of the procedures can create a schedule and calculate the objective value based on the actual completion times of the jobs.

Each of these two heuristics can be combined with the improvement procedure described in the next section. Also, a preprocessing problem reduction step is first performed before starting any of the procedures. This step is described first.

3.1. Preprocessing problem reduction

This step attempts to find jobs that will not be tardy no matter where they are scheduled, and hence will not contribute to the objective. These jobs are removed from the problem when the sequencing heuristics are applied. After the other jobs have been sequenced and scheduled, the jobs that were removed in this step will be placed at the end of the sequence and scheduled.

Let $P = \sum_{j=1}^{n} p_j$, let $A_j = ((P - p_j)/M) + p_j$. Azizoglu and Kirca (1998), show that a job j's maximum completion time will be A_j, and hence that job will not be tardy if $d_j \geq A_j$. This result is used in this step to identify jobs that will not be tardy, and therefore can be scheduled at the end. In the following procedure EDD[k] is the kth job if the jobs are sorted in non-descending order of the due dates, and ct is the total of the processing times of the jobs that have not been eliminated.

3.1.1. *Procedure problem reduction pseudo code*

```
Step 0. Input the number of machines (M) and number of jobs (n); the processing times and due dates of the jobs. Set $ct = P$, set $k = n$.
Step 1. If $d_{EDD[k]} \geq ((ct - p_{EDD[k]})/M) + p_{EDD[k]}$ then step 2, else step 3.
Step 2. Eliminate job EDD[k] from the problem. Set $ct = ct - p_{EDD[k]}$.
Step 3. Set $k = k - 1$. If $k > 0$ then step 1, else step 4.
Step 4. Stop. Output the list of jobs that will not be tardy if placed at the end.
```

In the above procedure step 1 does the comparison that determines if a job's due date is later than an upper bound on its completion time. If the comparison is successful step 2 eliminates the job from the problem and updates the total processing time of the remaining jobs. In the above procedure the jobs need to be sorted in EDD order, which takes $O(n * LN(n))$ time, and then pass through the jobs so the complexity of the algorithm is $O(n * LN(n))$.

3.2. *QB procedure*

This procedure attempts to obtain the advantages of the single machine heuristic QBackv6 developed by Valente and Schaller (2012), by adapting it to the parallel machines problem. This procedure works backward from the end of the schedule to the beginning. The QBackv6 procedure is described next. Let t^B denote the current time, that is, the completion time of the next job to be scheduled. The slack of job j is defined as $s_j^B = t^B - d_j$. Also, let p_{max} denote the maximum processing time of the currently unscheduled jobs, and T^{min} equal the minimum tardiness that could be incurred by scheduling one of the unscheduled jobs at the current position. Furthermore, let p equal the average of the processing times of the unscheduled jobs, and s^B equal the average of the s_j^B values for the unscheduled jobs. Finally, p_j^{mod} is set equal to $\min\{p_j, T^{min}\}$, while v is a parameter.

This dispatching rule schedules an early job whenever such a job is available. These jobs will not be tardy, and will decrease the tardiness of jobs that would be tardy. This is accomplished by giving early jobs a priority value that is positive (p_j), and tardy jobs a negative priority value.

If all of the jobs that still need to be scheduled will be tardy, then the dispatching rule considers each job's weight and the tardiness of the job if scheduled now, and possibly the job's modified processing time. By considering the weight and current tardiness of job j, the heuristic takes into account the contribution of the job to the objective value, if the job is scheduled now, allowing the procedure to see the reduction if the job is postponed until a later iteration. On the other hand, considering the processing time of job j provides a gauge on the impact in the other jobs' tardiness if job j is chosen now. The following index is used in this case: $-(w_j/p_j^{mod})[(s_j^B)^2 - v(\max\{t^B - p_{max} - d_j, 0\})^2]$. So, the priority index QB_j used in this rule to choose the next job to schedule becomes:

$$QB_j = \begin{cases} p_j & \text{if } s_j^B \leqslant 0 \\ -(w_j/p_j^{mod})[(s_j^B)^2 - v(\max\{t^B - p_{max} - d_j, 0\})2] & \text{otherwise} \end{cases},$$

where v equals

$$v = \begin{cases} 0 & \text{if } p \geqslant s^B \\ 1 & \text{if } p < s^B \,\&\, s^B/t^B > 0.5 \\ (s^B - p)/s^B & \text{if } p < s^B \,\&\, s^B/t^B \leqslant 0.5 \end{cases}$$

In order to adapt the above procedure to the parallel machine case we create an artificial single machine problem by calculating an artificial processing time p'_j for each job j. This artificial processing time is calculated as $p'_j = p_j/M$, where p_j is the processing time of job j and M is the number of machines. The values of the p'_j s are summed to obtain the initial current time. The procedure then uses the QBackv6 procedure (priority index) to solve the artificial single machine problem using the artificial processing times to create a sequence. So the p'_j s are substituted for p_js, p'_{max} is substituted for p_{max}, p'^{mod}_j is substituted for p_j^{mod} and p equals the average of the artificial processing times of the unscheduled jobs (p'_j). Once the sequence has been found, a schedule is created by working forward from the first job in the sequence to the last, assigning each job to the first available machine. In this step the actual processing times for each job are used. When this is completed an objective value is obtained. The pseudo-code for QB procedure is given below. Let U be the set of unscheduled jobs and let [j] be the job for the jth position of the sequence.

3.2.1. *QB Pseudo-code*

```
Step 0. Input the number of machines (M) and number of jobs (n); the processing times and due dates of the jobs.
Step 1. Initialization:
  1.1 For j: = 1 to n
    Set $p'_j = p_j/M$.
  1.2 Set U to consist of the set of jobs to be scheduled.
  1.3 Set $t^B = \sum_{j=1}^{n} p'_j$.
  1.4 Set $k = n$.
Step 2. Do While $k > 0$
  2.1 For $j \in U$
    Calculate index for job j.
  2.2 Select [k] using indexes.
  2.3 Set $t^B = t^B - p'_{[k]}$.
  2.4 Remove job [k] from set U.
  2.5 Set $k = k - 1$.
Step 3. Create schedule, calculate weighted squared tardiness and set $O_{QB}$ equal to the weighted squared tardiness.
Step 4. Stop. Output the best sequence, schedule and objective value.
```

Step 1 initializes the procedure by calculating the artificial processing times for each job and the initial time for t^B. Step 2 is performed n times. Each time step 2 is performed a job is selected for a position in the sequence to be developed, and t^B is updated. Step 3 develops the schedule by assigning jobs in sequential order to the first available machine, and calculates the total weighted squared tardiness. Step 4 stops the procedure. In the above procedure one job is added to the sequence during step 2 which is performed n times and the indexes have to be calculated for each unscheduled job so the complexity of the algorithm is $O(n^2)$.

3.3. *QBP procedure*

The QBP procedure also works backward from the end of the schedule to the beginning, and uses the index from the QBackv6 heuristic for single machine scheduling, like in the QB procedure. However, it uses the actual processing times (p_j) to create an estimated start and end time for each job. Let P be the total processing time of all the jobs that remain to be sequenced after the initial preprocessing step. The initial completion times of the first $(M - 1)$ machines are set equal to the integer portion of P/M. The completion time of machine M is set to P – the sum of the completion times of the other $(M - 1)$ machines. The current time (t^B) is set equal to the maximum of the machine completion times, and the machine that this time occurs on is the current machine (in the case of ties the lowest number machine is selected as the current machine).

Indexes are calculated for all the unscheduled jobs using the current time and the QBackv6 index. The job with the largest index is selected and placed in the sequence. The end time for the selected job is set equal to t^B and the estimated start time is set equal to $t^B - p_j$. This also becomes the current machine's updated completion time. When all the jobs are sequenced, a second sequence is created by sorting the jobs in earliest estimated start time order. Each of the sequences is evaluated by scheduling the jobs (working from the beginning of the sequence to the end) on the first machine that becomes available. The sequence that results in the best objective value is selected and its objective value becomes the objective value for the heuristic. The QBP priority index QBP_j follows; this index is the same as the QB_j index, except that the actual processing times (p_js) are used instead of the modified times (p'_js):

$$QBP_j = \begin{cases} p_j & \text{if } s_j^B \leqslant 0 \\ -(w_j/p_j\text{mod})[(s_j^B)^2 - v(\max\{t^B - p_{\max} d_j, 0\})^2] & \text{otherwise} \end{cases},$$

where v equals

$$v = \begin{cases} 0 & \text{if } p \geqslant s^B \\ 1 & \text{if } p < s^B \,\&\, s^B/t^B > 0.5 \\ (s^B - p)/s^B & \text{if } p < s^B \,\&\, s^B/t^B \leqslant 0.5 \end{cases}$$

The pseudo-code for QBP procedure is given below. Let U be the set of unscheduled jobs. Let [j] be the job for the jth position of the sequence, t_m be the estimated current time for machine m (m = 1, …, M), cm be the current machine and S_j be the estimated start time for job j. Finally, O_{QMP} is the objective value of the solution found by the procedure.

3.3.1. QBP Pseudo-code

```
Step 0. Input the number of machines (M) and number of jobs (n); the processing times and due dates of the jobs.
Step 1. Initialization:
  1.1 Set $P = \sum_{j=1}^{n} p_j$.
  1.2 Set U to consist of the set of jobs to be scheduled.
  1.3For m = 1 to M − 1
    Set $t_m = P/M$.
  1.4 Set $t_M = P - \sum_{m=1}^{M-1} t_m$.
  1.5 Set k = n.
Step 2. Do While k > 0
  2.1 Set cm = m with maximum $t_m$.
  2.2 Set $t^B = t_{cm}$.
  2.3 For j ∊ U
    Calculate index for job j.
  2.4 Select [k] using indexes.
  2.5 Set $S_{[k]} = t_{cm} - p_{[k]}$.
  2.6 Set $t_{cm} = t_{cm} - p_{[k]}$.
  2.7 Remove job [k] from set U.
  2.8 Set k = k − 1.
Step 3. Create schedule, calculate weighted squared tardiness and set $O_{QBP}$ equal to the weighted squared tardiness.
Step 4. Create a second sequence by sorting jobs in the non-descending order of their $S_j$ times.
Step 5. Create schedule, calculate weighted squared tardiness and if < $O_{QBP}$ then set $O_{QBP}$ equal to the weighted squared tardiness.
Step 6. Stop. Output the best sequence, schedule and objective value.
```

Step 1 initializes the procedure by calculating the estimated completion times for each machine. Step 2 is performed n times. Each time step 2 is performed the current machine is determined by finding the machine with the maximum completion time (ties are broken by selecting the machine with the lowest index). Also, a job is selected for a position in the sequence to be developed, t_{cm} is updated and the selected job's estimated start time ($S_{[k]}$) is calculated. Step 3 develops the schedule by assigning jobs in sequential order to the first available machine, and calculates the total weighted squared tardiness (O_{QMP}). Step 4 creates a second sequence by sorting the jobs' start times in non-descending order. Step 5 then creates a schedule for this sequence, calculates its weighted squared tardiness and, if it is less than that of the first sequence, the second sequence is selected as the solution and the objective is updated. Step 6 stops the procedure. In the above procedure one job is added to the sequence during step 2 which is performed n times and the indexes have to be calculated for each unscheduled job so the complexity of the algorithm is O (n^2).

4. Improvement procedure

An improvement procedure was developed which can be applied to a solution generated by either of the heuristics (QB and QBP). Two additional procedures are created by using the improvement procedure: QB_I and QBP_I.

The improvement procedure has three steps. In the first step, the procedure attempts to exchange jobs that are sequenced last on different machines. In the second step, a single-machine heuristic is used for the jobs assigned to each machine. The third step attempts to exchange pairs of jobs that are sequenced first on a machine. If an improved solution is found, the steps will be repeated.

4.1. Step 1 last jobs exchange procedure

This step attempts to exchange jobs that are scheduled last on machines to see if an improved objective value will result; since these jobs are sequenced last on a machine, switching a pair of these jobs will not affect other jobs. Note that the jobs identified by the preprocessing procedure described earlier are not considered here, because they will not be tardy.

There are some conditions that help to identify whether or not to perform an exchange. These conditions consider how the objective value changes as a job's completion time is increased one time unit, as first shown by Schaller and Valente (2012).

Each of the following conditions consider two schedules S and S′. Let S be a schedule with job j scheduled last on machine m, and job k scheduled last on machine n. Let b_j be the start time of job j on machine m and b_k be the start time of job k on machine n. Let $b_k < b_j$. Let C_j (S) be the completion time of job j in S and C_k (S) the completion time of job k in S. Let S′ be a schedule that is the same as S, except the positions of jobs j and k are exchanged in S′, so job k is scheduled last on machine m and job j is scheduled last on machine n. Then, we have $b'_j = b_k$ and $b'_k = b_j$. Let C_j (S′) be the completion time of job j in S′ and C_k (S′) the completion time of job k in S′. If one of the following properties is met the procedure will skip this candidate exchange, because it would not improve the objective value.

Condition 1: If $p_k > p_j$ & $d_k \le d_j$ & $w_k \ge w_j$, and jobs j and k are to be scheduled last on their respective machines, then schedule S will have an objective that is at least as good as the one resulting from schedule S′.

Condition 2: If $d_k \le d_j$, $w_k \le w_j$, $b_j + p_k - d_k \ge 0$, & $2 * w_k * (b_j + p_k - d_k) \ge 2 * w_j * (b_j + p_j - d_j)$, and jobs j and k are to be scheduled last on their respective machines, then schedule S will have an objective that is at least as good as the one resulting from schedule S′.

Condition 3: If $d_j \le d_k$, $w_k \ge w_j$, $C_k(S) - d_k \ge 0$, & $2 * w_k * (C_k(S) - d_k) > 2 * w_j * (b_k + p_j - d_j)$, and jobs j and k are to be scheduled last on their respective machines, then schedule S will have an objective that is at least as good as the one resulting from schedule S′.

The proofs for the three conditions are provided in Appendix A.

4.2. *Step 2 Single Machine Procedure*

In this step, for each machine, we sequence the jobs that are assigned to that machine using QBackv6, to see if an improved sequence is obtained on that machine.

4.3. *Step 3 First Jobs Exchange Procedure*

This step attempts to exchange jobs that are scheduled first on machines to see if there is an improvement. Since these jobs are sequenced first on a machine, switching a pair of these jobs will not affect the weighted squared tardiness of the pair of jobs, but could affect the weighted squared tardiness of the jobs that were sequenced after these jobs on the pair of machines. If the two jobs to be exchanged have the same processing time then exchanging the jobs will not affect the weighted squared tardiness of the jobs sequenced after them, and therefore this exchange does not need to be checked. If the two jobs to be exchanged have different processing times then, by exchanging the two jobs, the jobs that were previously sequenced on a machine after the job with the longer processing time could have their tardiness reduced, and hence have a lower weighted squared tardiness. However, the jobs that were originally sequenced on a machine after the job with the shorter processing time now will have later completion times, so their tardiness could increase and hence have a higher weighted squared tardiness.

Suppose a job j is completed at time t in a schedule. If job j were to be completed one time unit later (t + 1) in a different schedule then its contribution to the objective value would increase by 0 if $d_j > t$, and would increase by $w_j * (2 * T_j + 1)$ if $t \geq d_j$. Let $D_j(t) = 0$ if $d_j > t$ and $w_j * (2 * T_j + 1)$ if $t \geq d_j$. Let k and l be two machines and n_k and n_l be the number of jobs scheduled on machines k and l respectively. Let [j]k be the job sequenced in position j on machine k and let [j]l be the job sequenced in position j on machine l. Let $TD_m = \sum_{j=2}^{nm} D_{[j]m}(C_{[j]m})$.

When deciding whether to exchange the first jobs on machines k and l, the following two conditions are checked, and the exchange is attempted if either is met: (1) $p_{[1]k} > p_{[1]l}$ and $TD_k > TD_l$ or (2) $p_{[1]l} > p_{[1]k}$ and $TD_l > TD_k$. These conditions are checked because they are the most likely to result in a decrease in the objective value, but there is no guarantee that an exchange will result in a lower objective value. So a trial exchange is performed and if the total weighted squared tardiness is decreased then the exchange is implemented, otherwise the exchange is reversed.

4.4. *Improvement Procedure Pseudo-code*

The pseudo-code for the improvement procedure is given below. Let ISF be an indicator that specifies whether or not an improved solution was found.

4.4.1. *Improvement Pseudo-code*

```
Step 1. Initialization.
    1.1 Input the number of machines (M) and number of jobs (n); the processing times and due dates of the jobs.
    1.2 Input current solution and objective value.
    1.3 Set ISF = 1.
Step 2. Do while ISF = 1.
    2.1 Set ISF = 0.
    2.2 Perform last jobs exchange procedure. If an improved solution is found set ISF = 1.
    2.3 For m = 1 to M do
    Perform Qbackv6 procedure. If an improved solution is found set ISF = 1.
    2.4 Perform first jobs exchange procedure. If an improved solution is found set ISF = 1.
Step 3. Stop. Output the best sequence, schedule and objective value.
```

Step 1 initializes the procedure. In step 2 each of the sub-procedures is performed. If an improved solution is found the step is repeated. Step 3 stops the procedure. In steps 2.2 and 2.4 of the above procedure $O(M^2)$ exchanges are considered. Step 2.3 performs the Qbackv6 procedure once for each machine (M times) and the Qbackv6 procedure has $O(n^2)$ complexity so the complexity of step 2.3 is $O(n^2 * M)$ which will be less than $O(n^3)$ since M is usually less than n (if not the problem is trivial as each job can be assigned to a separate machine to minimize the objective).

5. Computational tests and results

This section describes the test of the proposed procedures and presents the results. We first describe some procedures that were used for comparison. Then, the test problems and the measure of performance are described. Finally, we present the results.

5.1. *Comparison procedures*

We include two very efficient procedures in the test for comparison purposes. These are described next.

5.1.1. *COM procedure*

This procedure is a combination of three simple and very efficient dispatching rules. Since it combines the three rules, we refer to this procedure as the combination procedure or by the abbreviation COM. The procedure develops three separate sequences using three rules. The resulting sequences are each evaluated to obtain an objective value. This is done for each sequence by assigning jobs (in the order of the sequence) to the first available machine. The sequence with the best objective value is selected.

The three rules that are used are Earliest Due Date (EDD), Longest Processing Time (LPT), and Shortest Processing Time (SPT). The rules are applied by performing a simple sort to develop the sequence. We also use the preprocessing problem reduction procedure, described in Section 3.1, when applying each of the three rules. The EDD and SPT rules were selected because they have been shown to be effective in minimizing total tardiness. The LPT rule was selected because it was shown to be effective in minimizing makespan for identical parallel machines (Coffman & Sethi, 1976; Graham, 1969).

5.1.2. *QAR procedure*

We adapted the single machine QAR procedure to the parallel machines environment and refer to it in this paper by the same name (QAR). The adaptation is very straightforward. The priority index for the parallel procedure to be used in the test is the same as the single machine procedure. The only difference is that each time a job is selected the completion time of the machine the job was assigned to is updated, and then the procedure finds the minimum completion time among all the machines (because this will be the machine that the next job to be selected is assigned to) and uses that time as the current time to compute the priority indexes for the unscheduled jobs.

The single machine QAR procedure was proposed by Valente and Schaller (2012), and is an adaptation to a quadratic setting of the AR heuristic developed by Alidaee and Ramakrishnan (1996), for the weighted linear tardiness problem.

Each time a job is added to the sequence, the single machine QAR heuristic picks the job with the maximum value of the priority index $QAR_{j.}$ In this index, t is the current time (that is, the completion time of the last scheduled job), p is the average processing time of the currently unscheduled jobs, $s_j = d_j - t - p_j$ is the slack of job j and k is a user-defined parameter. The QARj index is then given by:

$$\mathrm{QAR}_j = \begin{cases} (w_j/p_j)*[\ \bar{p}+ 2*\max(\ t+\ p_j - d_j;0] & \text{if } s_j \leqslant 0 \\ (w_j/p_j)*[k\bar{p}/(k\bar{p} + s_j) & \text{otherwise} \end{cases}$$

The parameter k provides the QAR heuristic with a look toward future selections capability, as first described by Vepsalainen and Morton (1987), in the context of the ATC rule for the single machine weighted linear tardiness problem. The parameter k is used to determine the number of jobs that are about to become tardy. Multiplying the average processing time of the currently unscheduled jobs by the parameter k allows the priority index to take into account the number of jobs which will become tardy during the next selections.

The parallel machines version of the QAR heuristic also selects the job with the largest value of the same priority index. The main adaptation required consists in the fact that the t is now the time on the current machine. Each time a job is to be added to the schedule, the current machine is the one with the minimum completion time/earliest possible start time among all the machines (with ties broken by selecting the machine with the lowest number). The chosen job is then assigned to the current machine, and the completion time on this machine is updated, by adding the processing time of the scheduled job.

5.1.3. Procedures with improvement procedure

Two additional comparison procedures can be formed by adding the improvement procedure to the COM and QAR procedures to form COM_I and QAR_I.

5.2. Test problems

The tests used problems that included eight levels of number of jobs (n) and three levels of number of machines (M). The levels of numbers of jobs (n) were 15, 20, 25, 30, 40, 50, 75 and 100. The levels of number of machines (M) were 2, 6 and 10. All of the problems were generated using random values. For each job j, the processing time p_j was created using a uniform distribution [1, 50], and an integer weight w_j was generated from a uniform distribution [1, 10].

In order to develop due dates for each job, we calculate the average processing time per machine for each problem and define this value as *P*. *P* is calculated by summing the processing times of all jobs and dividing by the number of machines (M) in a problem rounded to the nearest integer. We also use two parameters to develop due dates: T the due date tardiness tightness factor and R the range of due dates.

For each job j, the due date d_j was generated randomly by using a uniform distribution $[P(1 - \mathrm{T} - \mathrm{R}/2), P(1 - \mathrm{T} + \mathrm{R}/2)]$. The due date tardiness tightness factor (T) indicates how tight on average the due dates are relative to the amount of processing that is required. If this factor is high it is likely that jobs will be tardy and the weighted squared tardiness will be high. The due date range factor (R) indicates whether or not there is a great deal of dispersion in the due dates. Four levels of tardiness factor T were tested: 0.00, 0.25, 0.50 and 0.75. Four levels of the range of due dates, R, were also tested: 0.25, 0.50, 0.75, and 1.00. This gives a total of 16 combinations of T and R and allows for a wide variety of conditions to be tested. This method of generating due dates is a common approach that has been used since early papers for scheduling with an objective that is a function of a job's tardiness (Ow & Morten, 1988; Potts & van Wassenhove, 1991; Ow & Morten, 1989), and is consistent with recent papers addressing the weighted squared tardiness objective.

For each problem size, that is a combination of n and M, and for each combination of T and R, there are 10 random instances. Therefore, a total of 160 instances were generated for each problem size of n and M. For each heuristic procedure and for each instance we recorded the objective value of the solution found. We also used the lagrangean relaxation procedure developed by Luh and Hoitomt (1993), to calculate a lower bound on the optimal objective function value, as will be described in the next subsection. Turbo Pascal was used to code the procedures, and the procedures were executed on a Dell Inspiron 1525 1.60 GHz Laptop computer.

5.3. Lower bound calculation using Luh and Hoitomt (1993)'s lagrangian relaxation procedure

An existing procedure developed by Luh and Hoitomt (1993), was used to calculate a lower bound. This procedure uses Lagrangian relaxation. The machine capacity constraint of the machines is relaxed so, at a given point of time, the number of jobs processed could be greater than the number of machines, thereby providing a lower bound for the objective. The problem can be written as the integer program that follows. In the integer program K is an upper bound on the completion time of the last job completed on any machine and δ_{jk} indicates if job j is active during time unit k.

$$\text{Minimize } Z = \sum_{j=1}^{n} w_j^* T_j^2$$

$$\text{Subject to: } \sum_{j=1}^{n} \delta_{jk} \geqslant M \quad \text{for } \ k = 1,\ldots,K \tag{1}$$

Luh and Hoitomt relax constraint set (1) to form the relaxed problem using Π_k as the multipliers for the machine capacity constraint:

$$\text{Minimize} \sum_{j=1}^{n} w_j * T_j^2 + \sum_{k=1}^{K} \left(\Pi_k^* \left(\sum_{j=1}^{n} \delta_{jk} - M \right) \right)$$

This relaxation decouples the problem into n subproblems, one for each job, which can be solved by finding the starting time that minimizes $w_j * T_j^2 + \sum_{k=1}^{K} (\Pi_k * \delta_{jk})$ for each job j.

Subgradient optimization is the method used to update the Lagrangian multipliers. The procedure performs for a fixed number of iterations. To calculate the lower bounds the procedure was run for a maximum of $n * \sqrt{n}$ iterations of the subgradient optimization. For additional information about this procedure see (Luh & Hoitomt, 1993). Each time an iteration of the procedure is performed, a feasible solution is also created by sorting the jobs, based on the starting times that the relaxation generated. A job schedule is created based on this sequence and the resulting objective value is calculated, which provides an upper bound.

5.4. Performance measure

We use the Relative Deviation Index (RDI) versus the lower bound as our measure of performance for the procedures to be compared. The Relative Deviation Index was first proposed by Zemel (1981) and is a commonly used performance measure for problems that include an objective that is a function of tardiness (see for example: Kim, 1993; Kim, Lim, & Park, 1996; Vallada, Ruiz, & Minella, 2008). In order to define RDI let O_H be the objective value generated by heuristic H for the problem, O_W be the worst objective value generated for the problem by the heuristics to be compared and LB be the lower bound described in the previous section. The RDI for a problem is defined as: $(O_H - \mathrm{LB})/(O_W - \mathrm{LB})$.

There will be problems in which the weighted squared tardiness of the worst solution generated for a problem, as well as the lower bound for that problem, are equal to 0, and the previous equation would be undefined because the denominator is equal to 0. In these cases, the RDI for all the heuristics to be compared is set to zero for that problem (to avoid dividing by zero), because they all generated an optimal solution and hence there was no error. The RDI for a heuristic will return a value between 0 and 1. Values closer to 0 indicate better performance and values closer to 1 indicate worse performance.

Table 1
Average Relative Deviation Index (RDI) for the dispatching procedures.

# of Machines	# of Jobs	Procedure			
		COM	QAR	QB	QBP
2	15	0.62	0.30	0.24	0.11
	20	0.57	0.24	0.16	0.09
	25	0.57	0.21	0.12	0.07
	30	0.58	0.19	0.12	0.06
	40	0.58	0.21	0.12	0.07
	50	0.56	0.17	0.08	0.05
	75	0.57	0.16	0.07	0.05
	100	0.57	0.15	0.09	0.04
6	15	0.69	0.31	0.67	0.22
	20	0.63	0.26	0.61	0.13
	25	0.62	0.30	0.50	0.13
	30	0.61	0.27	0.43	0.10
	40	0.59	0.25	0.33	0.07
	50	0.57	0.23	0.26	0.06
	75	0.55	0.20	0.15	0.04
	100	0.55	0.18	0.12	0.03
10	15	0.55	0.32	0.85	0.22
	20	0.67	0.31	0.78	0.28
	25	0.67	0.29	0.72	0.18
	30	0.65	0.30	0.67	0.17
	40	0.60	0.24	0.52	0.11
	50	0.61	0.27	0.47	0.09
	75	0.56	0.24	0.30	0.06
	100	0.55	0.23	0.23	0.04

5.5. Results

In this section the results of the tests are presented. First, the results for the dispatching heuristics without the improvement procedure are presented. Then, the heuristics that incorporate the improvement procedure are compared.

5.5.1. Dispatching procedures' results

Table 1 shows the results for the dispatching procedures. For each procedure, the results are shown by the number of machines and number of jobs, and report the average relative deviation index.

The results show that the QBP procedure had the lowest RDI for all of the combinations of number of machines (M) and number of jobs (n) and the COM procedure had the highest RDI for all but four of the combinations of M and n. The QB procedure had the second lowest RDI and the QAR procedure the third lowest RDI when the number of machines is 2. The QAR and QB procedures are second and third when the number of machines is 6 or 10 but the order of the procedures varies, with QAR performing better when the ratio of jobs to machines (number of jobs/number of machines) is low, and the QB procedure performing better when the ratio is higher.

These results show the advantages of trying to work backward from the end of the schedule to the beginning (QB and QBP procedures), as well as using more sophisticated indices (QAR, QB and QBP procedures, as opposed to the COM procedure), for selecting a job when working on a position in a sequence. The QBP procedure uses a more sophisticated method of working backward and estimating job completion times, and consistently outperformed the other procedures.

Tests were conducted to determine if the differences between these four procedures are statistically significant. The procedures were used on the same instances, so a paired-samples test should be employed. Since the assumptions of either the paired-samples *t*-test, or ANOVA with repeated measures, were not met, the non-parametric Wilcoxon signed-rank test was chosen.

This test was applied to each pair of procedures (6 pairs in total), and for each combination of number of machines and number of jobs (24 such combinations), giving a total of 144 comparisons. The significance level was set at 0.05. Given that multiple tests were performed, Holm's sequential Bonferroni procedure was applied to adjust the significance level, thus taking into account the multiple comparisons.

These tests showed that the differences between procedures are significant. Indeed, the hypothesis of identical performance was not rejected in only 5 of the 144 tests performed, both concerning the comparison between the QAR and QB procedures. More detailed information about the tests' results is given in the electronic supplementary material.

Table 2
Average Relative Deviation Index (RDI) for the dispatching procedures by due date tightness and range parameter for n = 50 and M = 6.

Due Date Parameters		Procedure			
T	R	COM	QAR	QB	QBP
0.00	0.25	0.000	0.000	0.000	0.000
0.00	0.50	0.000	0.000	0.000	0.000
0.00	0.75	0.000	0.000	0.000	0.000
0.00	1.00	0.000	0.000	0.000	0.000
0.25	0.25	0.997	0.434	0.592	0.113
0.25	0.50	0.292	0.689	0.962	0.093
0.25	0.75	0.044	0.298	0.890	0.008
0.25	1.00	0.00	0.05	0.20	0.002
0.50	0.25	1.00	0.109	0.096	0.035
0.50	0.50	1.00	0.341	0.181	0.082
0.50	0.75	0.959	0.756	0.450	0.185
0.50	1.00	0.820	0.828	0.650	0.236
0.75	0.25	1.00	0.049	0.038	0.023
0.75	0.50	1.00	0.055	0.041	0.035
0.75	0.75	1.00	0.056	0.057	0.054
0.75	1.00	1.00	0.084	0.063	0.058

Table 2 shows the results for the dispatching heuristics by the due date tightness and range parameters when the number of jobs was 50 and the number of machines was 6. For each procedure the results are shown by the due date tightness and range parameters, and report the average relative deviation index for each procedure.

The results show that when T = 0.00 all of the procedures have an RDI equal to 0. When T = 0.00 the due dates are relatively loose and all of the procedures were able to find a solution with no tardy jobs and hence have a total weighted squared tardiness objective of 0. For the other sets of parameters the QBP procedure is consistently very good and is best among the procedures, with the exception of T = 0.25 and R = 1.00 in which it is second best. The other procedures are less consistent across the parameters. For other combinations of n and M the results are generally consistent with those shown in Table 2. When T = 0.00 all of the procedures were able to generate solutions with no tardy jobs when the ratio of jobs to machines was greater than 7.

These results show that the QBP heuristic is an effective procedure for the problem and is consistent across a wide variety of conditions. It is recommended for large scale problems that need to be solved quickly.

5.5.2. Dispatching procedures with the improvement procedure results

In this section we compare the results of the dispatching procedures with the inclusion of the improvement procedure described in Section 3. First we compare each procedure with the improvement included to its counterpart without the improvement procedure. To do this, we looked at the percentage reduction in the gap between the objective obtained by a procedure for a problem and the problem's lower bound. We refer to this measure as GAPRED% and to calculate it we use the following. O_H is the objective value generated by heuristic H without the improvement procedure, O_{H_I} is objective value generated by heuristic H with the improvement procedure included, and LB is the lower bound for the problem. Then, GAPRED% is defined as GAPRED% $= 100 - (O_{H\_I} - LB)/(O_H - LB) * 100$ if $O_H - LB > 0$, otherwise the GAPRED% is set to 0. Table 3 shows the results. For each combination of n and M, the table shows the average percentage reduction in

Table 3
Average percentage GAP reduction (GAPRED%) when the improvement procedure improvement procedure is included.

# of Machines	# of Jobs	Procedure			
		COM_I	QAR_I	QB_I	QBP_I
2	15	43.98	28.94	17.74	6.90
	20	44.81	29.24	13.15	3.94
	25	45.87	32.32	10.56	2.46
	30	46.67	32.01	9.68	3.03
	40	49.03	33.29	8.54	3.47
	50	48.67	34.64	7.43	2.79
	75	50.24	36.29	6.86	2.49
	100	51.56	37.01	7.31	1.99
6	15	62.27	53.69	62.48	42.25
	20	62.17	52.16	58.60	34.93
	25	54.63	46.98	47.16	28.87
	30	69.54	59.43	67.21	50.48
	40	62.83	53.60	54.58	38.53
	50	59.95	49.90	46.57	35.52
	75	53.95	45.46	35.97	21.83
	100	50.34	42.99	29.70	18.70
10	15	67.58	61.42	77.98	44.81
	20	76.51	61.32	73.60	55.86
	25	78.11	66.79	72.76	56.22
	30	52.36	45.73	42.50	24.94
	40	49.85	42.54	32.90	18.57
	50	50.34	42.42	27.95	14.20
	75	48.04	40.85	23.09	9.54
	100	48.57	39.16	20.54	6.86

Table 4
Average Relative Deviation Index (RDI) for the dispatching procedures with the improvement procedure.

# of Machines	# of Jobs	Procedure			
		COM_I	QAR_I	QB_I	QBP_I
2	15	0.63	0.41	0.40	0.19
	20	0.58	0.36	0.36	0.29
	25	0.59	0.38	0.35	0.28
	30	0.59	0.34	0.33	0.26
	40	0.60	0.34	0.32	0.26
	50	0.59	0.33	0.28	0.25
	75	0.60	0.31	0.28	0.25
	100	0.60	0.30	0.25	0.23
6	15	0.64	0.42	0.63	0.40
	20	0.62	0.39	0.60	0.29
	25	0.61	0.40	0.54	0.26
	30	0.60	0.39	0.45	0.21
	40	0.60	0.36	0.44	0.20
	50	0.58	0.33	0.35	0.18
	75	0.58	0.29	0.26	0.15
	100	0.59	0.28	0.22	0.16
10	15	0.55	0.54	0.74	0.50
	20	0.62	0.51	0.77	0.47
	25	0.64	0.46	0.74	0.35
	30	0.65	0.45	0.62	0.28
	40	0.63	0.42	0.55	0.26
	50	0.59	0.41	0.53	0.22
	75	0.58	0.36	0.39	0.18
	100	0.58	0.36	0.31	0.15

the gap obtained by including the improvement procedure in each of the procedures.

The results show that including the improvement procedure helps all of the procedures. The results also show that the improvement procedure has a bigger impact on poorer performing procedures. For example, including the improvement procedure with the COM procedure, the procedure that performed the poorest, resulted in the largest percentage reduction in the gap and including the improvement procedure in the QBP procedure, which performed best, resulted in the lowest percentage reduction in the gap. Part of the reason for this is the better a procedure is, the more likely it is to generate optimal solutions, and therefore including the improvement procedure will not help. For example, when $T = 0.00$, $n > 20$, and $M = 2$ all of the procedures generated solutions with an objective value equal to 0, which is optimal, and therefore the improvement procedure could not help.

We conducted a statistical test to determine if the improvement given by the improvement procedure is statistically significant. Again, we used the Wilcoxon signed-rank test, with a significance level of 0.05. This test was applied to all pairs of procedures with and without the improvement procedure (that is., COM_I vs COM, QAR_I vs QAR, …), and for each combination of number of machines and number of jobs. Holm's procedure was again used to correct for the multiple comparisons. All these comparisons were statistically significant. Therefore, the procedures that include the improvement procedure are significantly better.

We also compared the procedures with the improvement procedure included with each other, using the relative deviation index (RDI). These results are shown in Table 4.

The results shown in Table 4 show that although the performance of the procedures is closer together, the QBP_I procedure is the best performing procedure for all the combinations of n and M, and the COM_I procedure was the worst performing procedure for all but three of the combinations of n and M (it was second worst for these three). These results show the advantage of using the QBP procedure for the problem.

Statistical tests were also performed to compare the procedures that include the improvement procedure among themselves. As before, the Wilcoxon signed-rank test was applied, with a significance level of 0.05. The test was applied to each pair of procedures (6 in total), and for each of the 24 combinations of number of machines and number of jobs. As usual, Holm's sequential Bonferroni was used to correct the significance level in order to take into account the multiple comparisons.

The tests showed a statistically significant difference in 118 of the 144 comparisons. Of the 26 comparisons where a significant difference was not found, 11 correspond to comparisons between QAR_I and QB_I, while 9 occurred in the comparison between QAR_I and QBP_I. So, there was a not a statistically significant difference between QAR_I and QB_I (QAR_I and QBP_I) in about half (40%) of the cases. More detailed information about the tests' results is given in the electronic supplementary material.

All of the procedures considered are able to generate solutions very quickly. Each procedure averaged less than 0.10 s per problem for all of the problem sizes (combinations of n and M).

In order to provide a sense of the errors when using the two best performing procedures, QBP and QBP_I, we report the GAP% from the lower bound in table 5. The GAP% for a problem is defined as: $(O_H - LB)/O_H * 100$, where O_H is the objective value generated by heuristic H (QBP or QBP_I) and LB is the lower bound for the problem. If $O_H = 0$ then the GAP% for that heuristic is set equal to 0 for that

Table 5
Average Gap % from the lower bound for the QBP and QBP_I procedures.

# of Jobs	Procedure					
	QBP			QBP_I		
	M = 2	M = 6	M = 10	M = 2	M = 6	M = 10
15	35.40	41.60	29.70	14.97	15.04	4.04
20	31.72	40.44	35.32	12.45	15.11	12.04
25	31.97	36.99	42.86	12.51	12.88	14.33
30	12.94	17.25	24.44	12.81	13.10	12.38
40	11.87	12.53	18.92	11.72	9.31	10.77
50	12.04	11.25	17.95	11.96	9.53	10.32
75	10.79	8.45	12.89	10.72	7.79	8.67
100	11.07	8.13	11.20	11.04	8.13	6.19

problem (to avoid dividing by zero). These results show the GAP%s generally become smaller as the ratio of jobs to machines increases. Also, using the improvement does help reduce the larger error percentages and the procedure becomes much more consistent.

6. Incorporating the procedures in other procedures

The QBP and improvement procedures described in earlier sections can be incorporated into more computationally expensive procedures, and may result in better solutions. The procedures could be used to create initial solutions for procedures that require one, such as iterated local search, tabu search or simulated annealing procedures. The improvement procedure can be applied to solutions that are developed in procedures.

In this section, we demonstrate how using the QBP_I and improvement procedure with the existing lagrangean relaxation (LR) procedure developed by Luh and Hoitomt (1993) for the problem can result in improved solutions. We also show how an iterated local search procedure developed by Goncalves et al. (2016) for the single-machine weighted squared tardiness problem can be modified by using these proposed procedures.

6.1. Enhanced lagrangean relaxation based procedure

In Section 5.3 we described the lagrangean relaxation based procedure for developing lower bounds developed by Luh and Hoitomt (1993). Each time an iteration of the procedure is performed to develop a lower bound a feasible solution is also created to obtain an upper bound. This is accomplished by sorting the jobs, based on the starting times for each job that the relaxation generated. A job schedule is created based on this sequence and the resulting objective value is calculated which is an upper bound. The best upper bound found and its associated solution is retained. We refer to this existing procedure as LR.

We modified the procedure with two changes to create an enhanced LR procedure that we refer to as LRQ_I. First, the QBP_I procedure is run at the beginning of the LR procedure to obtain an initial solution. Also, each time a feasible solution is developed during an iteration of the LR procedure, the improvement procedure described earlier is applied.

6.2. Iterated local search procedure

Goncalves et al. (2016), developed an iterated local search for minimizing weighted squared tardiness in a single-machine. This iterated local search procedure and other meta-heuristics were tested, and the iterated local search generated solutions with better objective values. Here we adapt the iterated local search procedure to the parallel machine problem by incorporating the proposed procedures.

In the iterated local search developed by Goncalves et al. (2016), an initial solution is first generated, and then a local search is performed that may result in a better solution that becomes the current solution. Then a new solution is developed by modifying the current solution, by applying a series of random moves to this current solution. This is known as perturbing, or *kicking*, the current solution. This continues until a predetermined number of iterations are performed that do not result in an improved solution, in which case the procedure backtracks to the best solution found so far and the *kicking* and local search process is repeated. When a specified stopping procedure is met, or if the procedure finds a solution with a weighted squared tardiness of 0, the procedure terminates.

The pseudo-code for Goncalves et al. (2016), iterated local search procedure is given below. In this pseudo-code, (S_{best}, $of\ v_{best}$) denote the best solution found so far and its corresponding objective function value, respectively. Similarly, (S, $of\ v_S$) and (S_k, $of\ v_k$) provide the same information for the current solution and the kicked solution (that is, the solution obtained by performing a kick on the current solution), respectively.

6.2.1. Iterated local search pseudo-code

```
0. Input the number of machines (M) and number of jobs (n); the processing times and due dates of the jobs.
1. Set ($S_{best}$, $of\ v_{best}$) = ($\varnothing$,$\infty$).
2. ($S$, $of\ v_S$) = Generate_Initial_Solution().
3. Perform_Local_Search($S$).
4. If $of\ v_S < of\ v_{best}$, set ($S_{best}$, $of\ v_{best}$) = ($S$, $of\ v_S$).
5.While stop criterion is not met:
    5.1. ($S_k$, $of\ v_k$) = Perform_Kick($S$).
    5.2. Perform_Local_Search($S_k$).
    5.3. If $of\ v_k < of\ v_{best}$, set ($S_{best}$, $of\ v_{best}$) = ($S_k$, $of\ v_k$).
    5.4. If Perform_Backtrack() == TRUE, set ($S$, $of\ v_S$) = ($S_{best}$, $of\ v_{best}$).
    5.5. Else, set ($S$, $of\ v_S$) = ($S_k$, $of\ v_k$).
6. Output the best sequence, schedule and objective value.
```

Step 1 initializes the best solution found and the respective objective function value to an empty sequence and infinity, respectively. Step 2 generates the initial solution. Step 3 applies the local search to the initial solution. Step 4 updates the best solution if one is found. In step 5, the algorithm iterates until a stopping criterion is met.

During an iteration of step 5, a new solution S_k is obtained by executing a kick on the current solution S (step 5.1). A kick consists in performing α random swaps (the swapped jobs need not be adjacent), where α is a parameter selected by the user. Steps 5.2 and 5.3 are the performing the local search and then updating of the best solution found so far if necessary.

Finally, a new current solution is set in steps 5.4 and 5.5. If a so-called *backtrack* is to be performed, the current solution is set equal to the best solution found so far. Otherwise, the kicked solution becomes the new current solution. In the proposed implementation, backtracking is performed when β consecutive iterations have been performed without improving the best solution found so far, where β is also a user selected parameter.

In order to implement the above procedure for the parallel machines problem, we use the QBP procedure to generate the initial solution in step 2, and the improvement procedure described in section four as the local search in steps 3 and 5.2. We used the parameters selected by Goncalves et al. (2016), for α and β. These two parameters were set equal to 5. Goncalves et al. (2016), set a maximum run time as the stopping criteria and we use the same criteria. The maximum run time was set equal to $0.5 + 0.0001 * n^2$. This procedure is referred to as ILS in the remainder of the paper.

6.3. Computational test

We tested the three procedures (LR, LRQ_I and ILS) using the same test instances as described in Section 5. In order to have a fair comparison we allowed each procedure to run for the same amount of time on a problem and, if this time limit was reached, the procedure was terminated. As previously mentioned, the maximum run time (max_rt) for each procedure was set as: $maxrt = 0.5 + 0.0001 * n^2$. We used the relative deviation index and gap percentage, described in sections 5.3 and 5.5, as the measures of performance. Table 6 shows the results for these procedures. For each procedure the results are shown by the number of machines and number of jobs, and report the average relative deviation index and gap percentage for each procedure.

The results show that the ILS procedure was best when M = 2 for the GAP% for all levels of n and was the best for RDI when M = 2 and n > 25. The LRQ_I procedure was best when M = 6 or 10 for n > 15, with the exception of M = 6 and n = 100 in which ILS and LRQ_I are about equal. The results also show that including the enhancements in the lagrangean relaxation based procedure (LRQ_I) resulted in improved solutions compared to the original LR procedure. It appears that as the ratio of the number of jobs per machine (n/M) increases, the ILS

Table 6
Average Relative Deviation Index (RDI) and Average Gap Percentage (GAP%) for the lagrangean relaxation based procedures and the iterated local search procedure.

# of Machines	# of Jobs	Procedure					
		LR		LRQ_I		ILS	
		RDI	GAP%	RDI	GAP%	RDI	GAP%
2	15	0.61	12.98	0.60	12.73	0.64	12.65
	20	0.52	12.20	0.46	11.41	0.55	11.12
	25	0.57	13.53	0.47	11.86	0.50	11.46
	30	0.62	16.38	0.36	12.99	0.33	12.37
	40	0.63	20.72	0.15	11.62	0.14	11.37
	50	062	23.28	0.12	11.99	0.11	11.80
	75	0.63	26.92	0.09	10.78	0.08	10.54
	100	0.63	29.86	0.08	11.04	0.07	10.85
6	15	0.76	6.73	0.73	6.61	0.72	6.20
	20	0.73	5.86	0.68	5.52	0.79	8.16
	25	0.46	6.31	0.43	5.08	0.77	9.49
	30	0.29	7.24	0.26	6.63	0.68	11.60
	40	0.30	8.61	0.25	7.35	0.61	9.01
	50	0.37	10.25	0.28	6.96	0.54	8.59
	75	0.56	13.79	0.33	8.04	0.36	7.70
	100	0.62	17.88	0.14	8.12	0.14	8.06
10	15	0.72	2.28	0.70	2.23	0.76	2.21
	20	0.41	5.63	0.39	4.93	0.71	5.96
	25	0.30	5.44	0.26	4.23	0.69	6.11
	30	0.32	2.99	0.30	2.76	0.73	6.50
	40	0.28	5.36	0.23	5.07	0.68	8.04
	50	0.29	6.38	0.23	5.08	0.63	9.98
	75	0.30	8.66	0.22	5.65	0.56	8.47
	100	0.38	9.80	0.29	5.62	0.47	6.14

procedure improves relative to the lagrangian relaxation based procedures, as evidenced by its better performance when the number of machines was two. The ILS procedure is recommended for medium sized problems when the ratio of jobs to machines is less than 15, and LRQ_I is recommended, when it is less than 15.

Once more, the Wilcoxon signed-rank test, with a significance level of 0.05, was used to compare these 3 procedures among themselves. This test was applied to each pair of procedures (3 such pairs), and for each of the 24 combinations of number of machines and number of jobs. The multiple comparisons were taken into account by applying Holm's sequential Bonferroni procedure.

The tests showed a statistically significant difference in all but two of the comparisons involving LR and ILS. In what considers both the LR vs LRQ_I and the LRQ_I vs ILS comparisons, respectively 20 and 17 of the 24 comparisons showed a statistically significant difference. More detailed information is given in the electronic supplementary material.

We used the branch-and-bound procedure developed by Schaller and Valente (2013) to obtain optimal solutions for the 15 job problems with 2, 6, and 10 machines. Table 7 shows the average GAP% for the best performing procedures when compared to the optimal objective values for the 15 job problems. The GAP% for a problem in this table is defined as: $(O_H - O_O)/O_H * 100$, where O_H is the objective value generated by heuristic H (QBP or QBP_I) and O_o is the optimal objective value for the problem. If $O_H = 0$ then the GAP% for that heuristic is set equal to 0 for that problem (to avoid dividing by zero).

The results show that the enhanced lagrangian relaxation procedure and the iterated search with the QBP and improvement procedures embedded provide results very close to optimal. The QBP procedure provided good results for 2 and 10 machines and with the improvement procedure provided better results.

Table 7
Average gap % from the optimal objective value for the 15 job problems.

Procedure	Number of Machines		
	2	6	10
QBP	5.78	23.29	6.58
QBP_I	4.48	10.14	2.26
LRQ_I	0.53	0.82	0.02
ILS	0.22	0.41	0.00

7. Conclusions and future research

This paper considers the identical parallel machines problem for the weighted squared tardiness objective. Two efficient procedures that can generate solutions for the problem very quickly were proposed. An improvement procedure that is very fast was also proposed, and this improvement procedure can be used with either of the heuristic procedures to create two additional procedures.

The proposed procedures, as well as two procedures that were used for the single-machine problem, used problems generated randomly to test their performance. Several levels of parameters were included in the test.

The test results showed that the QBP and QBP_I generated the best solutions and the QBP_I procedure is recommended for large scale problems.

For more moderate sized problems, it is practical to use the lagrangian relaxation procedure developed by Luh and Hoitmot, 1993, as well as metaheuristics such as iterated local search. The test results showed that QBP_I and the improvement procedure can be incorporated in lagrangian relaxation and iterated local search to generate better solutions.

Possible avenues for future research would include incorporating the use of subcontractors as an alternative for reducing the penalties for tardy deliveries. Also, other characteristics can be incorporated into the problem such as the existence of release dates for each job.

Appendix A. Proofs of the conditions in Section 4.1

The objective can be rewritten as $Z = \sum_{j=1}^{n} Z_j$, where $Z_j = w_j * T_j^2$. Z_j represents the contribution of job j to the objective value. Suppose a job j is completed at time t in a schedule. Suppose $C_j = t$ and let $\frac{dZ_j}{dt}$ be the first derivative of Z_j and $\frac{d^2Z_j}{dt^2}$ be the second derivative of Z_j. Then $\frac{dZ_j}{dt} = 0$ if $t < d_j$, $\frac{dZ_j}{dt} = 2 * w_j * T_j$ if $t > d_j$, and $\frac{dZ_j}{dt}$ is undefined for $t = d_j$. $\frac{d^2Z_j}{dt^2} = 0$ if $t < d_j$, $\frac{d^2Z_j}{dt^2} = 2 * w_j$ if $t > d_j$, and $\frac{d^2Z_j}{dt^2}$ is undefined for $t = d_j$.

Condition 1: If $p_k > p_j$ & $d_k \le d_j$ & $w_k \ge w_j$, and jobs j and k are to be scheduled last on their respective machines, then schedule S will have an objective that is at least as good as the one resulting from schedule S′.

Proof. Since the jobs just exchange machines, and are scheduled last on the machines in both schedules, none of the other jobs are affected by the change from schedule S to S′ (their completion times remain the same). Therefore, to prove this condition it must be shown that $Z_j (S) + Z_k (S) \le Z_j (S') + Z_k (S')$. Let $A = b_j - b_k$. Note that $C_j (S) = C_j (S') + A$, and $C_k (S') = C_k (S) + A$. Also, since $p_k > p_j$ then $C_k (S) > C_j (S')$ and $C_k (S') > C_j (S)$. There are six cases: (1) $b_j + p_k \le d_k$. (2) $b_k + p_k > d_k$ and $b_k + p_j > d_j$. (3) $b_k + p_k > d_k$ and $b_k + p_j \le d_j < b_j + p_j$. (4) $b_k + p_k > d_k$ and $b_j + p_j < d_j$. (5) $b_k + p_k \le d_k < b_j + p_k$ and $b_j + p_j \le d_j$. (6) $b_k + p_k \le d_k < b_j + p_k$ and $b_j + p_j > d_j$. $\square$

Case (1) $b_j + p_k \le d_k$. In this case, $Z_j (S) = Z_k (S) = Z_j (S') = Z_k (S') = 0$ and $Z_j (S) + Z_k (S) \le Z_j (S') + Z_k (S')$.

Case (2) $b_k + p_k > d_k$ and $b_k + p_j > d_j$. Since $p_k > p_j$, $d_k \le d_j$, $w_k \ge w_j$, then $\frac{dZ_k}{dt}$ $(t = C_k (S)) \ge \frac{dZ_j}{dt'}$ $(t' = C_j (S'))$. Also, the above conditions imply $\frac{d^2Z_k}{dt^2} \ge \frac{d^2Z_j}{dt^2}$, so $\frac{dZ_k}{dt+a} \ge \frac{dZ_j}{dt'+a}$ for $t = C_k (S)$, $t' = C_j$

(S'), and $1 \le a \le A$. Therefore, $Z_j(S) + Z_k(S) \le Z_j(S') + Z_k(S')$.

Case (3) $b_k + p_k > d_k$ and $b_k + p_j \le d_j < b_j + p_j$. In this case, $Z_j(S') = 0$. $\frac{dZ_k}{dt} \ge \frac{dZ_j}{dt}$ for $t > d_j$, therefore $Z_k(S') - Z_k(S) > Z_j(S)$ so $Z_j(S) + Z_k(S) \le Z_j(S') + Z_k(S')$.

Case (4) $b_k + p_k > d_k$ and $b_j + p_j < d_j$. In this case $Z_j(S) = Z_j(S') = 0$. Since $Z_k(S') > Z_k(S)$, $Z_j(S) + Z_k(S) \le Z_j(S') + Z_k(S')$.

Case (5) $b_k + p_k \le d_k < b_j + p_k$ and $b_k + p_j \le d_j$. In this case $Z_j(S) = Z_j(S') = Z_k(S) = 0$. Since $Z_k(S') > 0$, $Z_j(S) + Z_k(S) \le Z_j(S') + Z_k(S')$.

Case (6) $b_k + p_k \le d_k < b_j + p_k$ and $b_j + p_j > d_j \ge b_k + p_j$. In this case $Z_j(S') = Z_k(S) = 0$. $C_k(S') > C_j(S)$ and $d_k \le d_j$, $w_k \ge w_j$ implies $Z_k(S') > Z_j(S)$ therefore $Z_j(S) + Z_k(S) \le Z_j(S') + Z_k(S')$.//

Condition 2: If $d_k \le d_j$, $w_k \le w_j$, $b_j + p_k - d_k \ge 0$, & $2 * w_k * (b_j + p_k - d_k) \ge 2 * w_j * (b_j + p_j - d_j)$ and jobs j and k are to be scheduled last on their respective machines, then schedule S will have an objective that is at least as good as the one resulting from schedule S′.

Proof. As above, none of the other jobs are affected by the change from schedule S to S′, and it is enough to show that $Z_j(S) + Z_k(S) \le Z_j(S') + Z_k(S')$. Let $A = b_j - b_k$. Note that $C_j(S) = C_j(S') + A$ and $C_k(S') = C_k(S) + A$. Also note that $w_k \le w_j$ and $2 * w_k * (b_j + p_k - d_k) \ge 2 * w_j * (b_j + p_j - d_j)$ imply $T_k(S') \ge W_j * T_j(S)/W_k$. There are six cases: (1) $b_j + p_k = d_k$. (2) $b_j + p_k > d_k$ and $b_j + p_j < d_j$. (3) $b_k + p_k > d_k$ and $b_k + p_j \le d_j < b_j + p_j$. (4) $b_k + p_k > d_k$ and $b_j + p_j < d_j$. (5) $b_k + p_k \le d_k < b_j + p_k$ and $b_j + p_j \le d_j$. (6) $b_k + p_k > d_k$ and $b_j + p_j > d_j$. □

Case (1) $b_j + p_k = d_k$. In this case, $Z_j(S) = Z_k(S) = Z_j(S') = Z_k(S') = 0$ and $Z_j(S) + Z_k(S) \le Z_j(S') + Z_k(S')$.

Case (2) $b_j + p_k > d_k$ and $b_j + p_j < d_j$. In this case $Z_j(S) = Z_j(S') = 0$. $Z_k(S') > Z_k(S)$ so $Z_j(S) + Z_k(S) \le Z_j(S') + Z_k(S')$.

Case (3) $b_k + p_k > d_k$ and $b_k + p_j \le d_j < b_j + p_j$. In this case $Z_j(S') = 0$. $T_k(S) = T_k(S') - A \ge (W_j * T_j(S)/W_k) - A$. $Z_k(S') - Z_k(S) = 2 * W_k * (2 * A * T_k(S') + A^2) \ge 2 * (2 * A * W_j * T_j(S) + A^2) \ge Z_j(S)$ therefore $Z_k(S') > Z_k(S)$ so $Z_j(S) + Z_k(S) \le Z_j(S') + Z_k(S')$.

Case 4) $b_k + p_k > d_k$ and $b_j + p_j < d_j$. In this case $Z_j(S) = Z_j(S') = 0$. Since $Z_k(S') > Z_k(S)$, $Z_j(S) + Z_k(S) \le Z_j(S') + Z_k(S')$.

Case 5) $b_k + p_k \le d_k < b_j + p_k$ and $b_j + p_j \le d_j$. In this case $Z_j(S) = Z_j(S') = Z_k(S) = 0$. Since $Z_k(S') > 0$, $Z_j(S) + Z_k(S) \le Z_j(S') + Z_k(S')$.

Case 6) $b_k + p_k > d_k$ and $b_j + p_j > d_j$. In this case since $w_k \le w_j$ and $2 * w_k * (b_j + p_k - d_k) \ge 2 * w_j * (b_j + p_j - d_j)$ then $\frac{dZ_k}{dt + a} \ge \frac{dZ_j}{dt' + a}$ for $t = b_k + p_k$, $t' = b_k + p_j$, and $0 < a \le A$. Therefore $Z_k(S') - Z_k(S) \ge Z_j(S') - Z_j(S)$ so $Z_j(S) + Z_k(S) \le Z_j(S') + Z_k(S')$.//

Condition 3: If $d_j \le d_k$, $w_k \ge w_j$, $C_k(S) - d_k \ge 0$, & $2 * w_k * (C_k(S) - d_k) > 2 * w_j * (b_k + p_j - d_j)$, and jobs j and k are to be scheduled last on their respective machines, then schedule S will have an objective that is at least as good as the one resulting from schedule S′.

Proof. Again, it suffices to show that $Z_j(S) + Z_k(S) \le Z_j(S') + Z_k(S')$. Let $A = b_j - b_k$. Note that $C_j(S) = C_j(S') + A$, and $C_k(S') = C_k(S) + A$. There are two cases: (1) $b_k + p_j \le d_j$ and (2) $b_k + p_j > d_j$. □

Case (1) $b_k + p_j \le d_j$. In this case $Z_j(S') = 0$. $T_k(S) = T_k(S') - A$. $Z_k(S') - Z_k(S) = 2 * w_k * (T_k(S') - T_k(S)) = 2 * W_k * (2 * A * T_k(S') + A^2)$. Since $w_k \ge w_j$, $T_j(S) \le A$ then $Z_k(S') - Z_k(S) \ge Z_j(S)$ and $Z_j(S) + Z_k(S) \le Z_j(S') + Z_k(S')$.

Case (2) 2) $b_k + p_j > d_j$. In this case since $w_k \ge w_j$ and $2 * w_k * (C_k(S) - d_k) \ge 2 * w_j * (b_k + p_j - d_j)$ then $\frac{dZ_k}{dt + a} \ge \frac{dZ_j}{dt' + a}$ for $t = C_k(S)$, $t' = b_k + p_j$, and $0 < a \le A$. Therefore $Z_k(S') - Z_k(S) \ge Z_j(S') - Z_j(S)$ so $Z_j(S) + Z_k(S) \le Z_j(S') + Z_k(S')$.//

Appendix B. Supplementary material

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.cie.2018.03.036.

References

Alidaee, B., & Ramakrishnan, K. R. (1996). A computational experiment of COVERT-AU class of rules for single machine tardiness scheduling problem. *Computers & Industrial Engineering, 30*, 201–209.

Azizoglu, M., & Kirca, O. (1998). Tardiness minimization on parallel machines. *International Journal of Production Economics, 55*, 163–168.

Coffman, E. G. & Sethi, R. (1976). A generalized bound on LPT sequencing. In *Proceedings of the 1976 ACM SIGMETRICS conference on computer performance modeling measurement and evaluation* (pp. 306–310).

Fisher, M. L. (1973a). Optimal solution of scheduling problems using lagrangian multipliers: Part I. *Operations Research, 21*(5), 1114–1127.

Fisher, M. L. (1973b). Optimal solution of scheduling problems using lagrangian multipliers: Part II. *Operations Research, 21*(6), 294–317.

Garey, M. R., & Johnson, D. S. (1978). Strong NP-completeness results: Motivation, examples and implications. *Journal of the Association for Computing Machinery, 25*, 499–508.

Gonçalves, T. C., Valente, J. M., & Schaller, J. E. (2016). Meta heuristics for the single machine weighted quadratic tardiness problem. *Computers and Operations Research, 70*, 115–126.

Graham, R. L. (1969). Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics, 17*(2), 416–429.

Hoitomt, D. J., Luh, P. B., Max, E., & Pattipati, K. R. (1990). Scheduling jobs with simple precedence constraints on parallel machines. *IEEE Control Systems Magazine,* 34–40.

Kim, Y. D. (1993). Heuristics for flowshop scheduling problems minimizing mean tardiness. *Journal of Operational Research Society, 44*, 19–28.

Kim, Y. D., Lim, H. G., & Park, M. W. (1996). Search heuristics for a flowshop scheduling problem in a printed circuit board assembly process. *European Journal of Operational Research, 91*, 124–143.

Lawler, E. L. (1977). A "Pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. In P. L. Hammer Eljbhk, & G. L. Nemhauser (Eds.). *Annals of discrete mathematics* (pp. 331–342). Elsevier.

Lenstra, J. K., Rinnooy Kan, A. H. G., & Brucker, P. (1977). Complexity of machine scheduling problems. In P. L. Hammer Eljbhk, & G. L. Nemhauser (Eds.). *Annals of discrete mathematics* (pp. 343–362). Elsevier.

Luh, P. B., & Hoitomt, D. J. (1993). Scheduling of manufacturing systems using the lagrangian relaxation technique. *IEEE Transactions on Automatic Control, 38*, 1066–1079.

Mokotoff, E. (2001). Parallel machine scheduling problems: A survey. *Asia-Pacific Journal of Operational Research, 18*, 193–242.

Ow, P. S., & Morten, T. E. (1988). Filtered beam search in scheduling. *International Journal of Production Research, 26*(1), 35–62.

Ow, P. S., & Morten, T. E. (1989). The single-machine early tardy problem. *Management Science, 35*(2), 177–191.

Potts, C. N., & van Wassenhove, L. N. (1991). Single-machine tardiness sequencing heuristics. *IIE Transactions, 23*(4), 346–354.

Schaller, J. E., & Valente, J. M. S. (2012). Minimizing the weighted sum of squared tardiness on a single machine. *Computers and Operations Research, 39*(5), 919.

Schaller, J. E. & Valente, J. M. S. (2013). Minimizing the weighted sum of squared tardiness on identical parallel machines. *Presented at the 2013 decision sciences Institute's National Meeting, Baltimore, Maryland.*

Sen, T., Sulek, J. M., & Dileepan, P. (2003). Static scheduling research to minimize weighted and unweighted tardiness: A state-of-the-art survey. *International Journal of Production Economics, 83*(1), 1–12.

Sun, X., & Noble, J. S. (1999). An approach to job shop scheduling with sequence-dependent setups. *Journal of Manufacturing Systems, 18*, 416–430.

Sun, X., Noble, J. S., & Klein, C. M. (1999). Single-machine scheduling with sequence dependent setup to minimize total weighted squared tardiness. *IIE Transactions, 31*, 113–124.

Taguchi, G. (1986). *Introduction to quality engineering.* Tokyo, Japan: Asian Productivity Organization.

Thomalla, C. S. (2001). Job shop scheduling with alternative process plans. *International Journal of Production Economics, 74*, 125–134.

Vallada, V., Ruiz, R., & Minella, G. (2008). Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers & Operations Research, 35*, 1350–1373.

Valente, J. M., & Schaller, J. E. (2012). Dispatching heuristics for the single machine weighted quadratic tardiness scheduling problem. *Computers and Operations Research, 39*(9), 2223.

Vepsalainen, A. P. J., & Morton, T. E. (1987). Priority rules for job shops with weighted tardiness costs. *Management Science, 33*, 1035–1047.

Zemel, E. (1981). Measuring the quality of approximate solutions to zero–one programming problems. *Mathematics of Operations Research, 6*, 319–332.