Hybrid of the weighted minimum slack and shortest processing time dispatching rules for the total weighted tardiness single machine scheduling problem with availability constraints

Ibrahim H. Osman · Hocine Belouadah · Krzysztof Fleszar · Maysaa Saffar

Abstract The paper addresses the total weighted tardiness single-machine scheduling problem with machine availability constraints. Each machine can handle one job at a time and has a set of maintenance periods, during which the machine must be stopped for service. Each maintenance period lasts for a fixed interval of time and starts after a fixed interval of processing non-resumable jobs. A non-resumable job must be processed and completed outside the maintenance periods. The single-machine periodic maintenance problems with a total weighted tardiness objective appear to be a new class of problems for which, in this paper, a mixed integer linear programming formulation and a new composite dispatching rule based on a combination of the weighted minimum slack and shortest processing time rules are introduced. New data set instances with different characteristics are generated using existing benchmarks from the literature. Extensive computational results are reported to testify the power of the proposed dispatching-based rule heuristic by comparing it to other heuristics inspired from the literature and developed for this new problem. Finally the paper concludes with a summary and highlights further research directions.

1 Introduction

Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TWT</td>
<td>total weighted tardiness</td>
</tr>
<tr>
<td>SMSP</td>
<td>single machine scheduling problem</td>
</tr>
<tr>
<td>MA</td>
<td>machine availability</td>
</tr>
<tr>
<td>PM</td>
<td>periodic maintenance</td>
</tr>
<tr>
<td>MILP</td>
<td>mixed integer linear programming</td>
</tr>
<tr>
<td>DR</td>
<td>dispatching rule</td>
</tr>
<tr>
<td>DRH</td>
<td>dispatching rule heuristic</td>
</tr>
</tbody>
</table>

The Total Weighted Tardiness (TWT) Single-Machine Scheduling Problem (SMSP) has been a very active research and practice areas for many decades based on a basic assumption that the machine is available at all times. In modern manufacturing and service industries the all-time availability assumption may not be justifiable; due to the owner's interest in maximizing the utilization of expensive and bottleneck machines. As a consequence, decision makers recognized the importance of introducing a periodic maintenance schedule to extend the life and availability of machine, to avoid breakdown and loss of services, and to reduce building-up of inventories. For instance, OTIS Company has designed an intelligent predictive system which records the working condition of an elevator to decide on maintenance visits in order to minimize elevator downtime. Such groundbreaking optimization system is one of OTIS competitive advantages to boost revenues and enhance reputations [8]. Furthermore, the
importance of introducing preventive maintenance (PM) to increase machine availability (MA) is also highlighted in Raza et al. [20] who emphasized that a sound PM in a manufacturing environment results in reducing corrective maintenance cost and defective production cost as well as increasing the availability of production facility. Finally, Cassady & Kutanoglu [3] stressed the practical importance of integrating PM planning and production scheduling for which they developed an integrated mathematical model to minimize the total weighted completion time on a single machine. They reported improvement with average savings ranging from 2 to 20% depending on the nature of the manufacturing system. Not to mention real life applications in the airline industry where there are special repair stations (virtual single-machine) to perform different types of maintenance repairs (A, B or C) with varying processing times on each aircraft. The repair stations are limited in number worldwide, and each station requires regular maintenance to increase its utility to meet its continuous demands of on-time maintenance of aircrafts, otherwise a heavy penalty would be incurred equal to at least the lost income from a non-repaired aircraft. Another application in the airline industry would be an aircraft that must be serviced after a fixed total of flying mileage before it is allowed to resume operation; delays in maintenance of any flight would normally incur a huge penalty cost. It should be noted that machine availability constraints in general include maintenance, vacations, and breaks. In this paper, we limit our study to a set of maintenance periods during which the machine is not available. We denote the total weighted tardiness single machine scheduling problem with availability constraints as (TWT-SMSP-MA) and that with a periodic maintenance schedule by (TWT-SMSP-PM).

On the theoretical side, NP-hardness in the strong sense of TWT-SMSP was established by Lenstra, Rinnoy Kan and Bruker [14]. Adiri, Bruno, Frosting and Rinnoy Kan [1] considered the SMSP to minimize total completion time where the machine is unavailable during some intervals. They showed that SMSP is NP-hard when there is only one maintenance period in the schedule. Leon & Wu [15] introduced SMSP with the machine availability constraint to minimize the maximum lateness of jobs. They presented a nonlinear mixed integer model where the maintenance periods are treated as a set of additional jobs with flexible ready-times and artificial due dates. Lee [13] proved that SMSP-MA with one maintenance period to minimize the total tardiness is NP-hard. Consequently, TWT-SMSP-PM with \( m (m > 1) \) maintenance periods is also NP-hard since it is a generalization of both TWT-SMSP and SMSP-PM. Qi et al.[19] also proved that SMSP where jobs and maintenances are scheduled simultaneously to minimize the total completion time is NP-hard in the strong sense for which they developed three heuristics and a branch-and-bound algorithm for solving instances up to 20 jobs. Graves and Lee [10] presented SMSP with one maintenance activity to minimize the total weighted completion time.

Recently, Liao & Chen [16] considered SMSP-PM with several maintenance periods to minimize the maximum tardiness for which they developed a branch-and-bound method and a heuristic based on batching of jobs and earliest due date scheduling rule to solve instances up to 30 jobs. Chen [4] developed a heuristic and a branch-and-bound algorithm to minimize the total flow time in order to solve instances up to 30 jobs. Raza and Al-Turki [20] addressed the problem of joint scheduling of maintenance operations and jobs to minimize the total completion time to ensure high productivity. They discussed properties for an optimal schedule in order to develop tabu search and simulated annealing meta-heuristics to solve instances up to 40 jobs. Ji et al. [11] developed a construction heuristic with a worst-case ratio less than two to minimize the makespan. Chen [5] proposed a heuristic and a branch-and-bound algorithm to minimize separately the total flow time and the maximum tardiness. Chen [6] developed a heuristic as well as a branch-and-bound algorithm considered to minimize the number of tardy jobs and solved instances up to 32 jobs.

From the above review, it can be seen that SMSP-PM with the objective of minimizing total weighted tardiness, has not been addressed in the literature. Moreover, jobs with due dates are of great interest in business since the cost of tardiness incurs different types of penalties such as: dissatisfied customers with loss of future sales and goodwill, and rush shipping cost among others, Biskup and Feldmann [2]. Hence, our aim in this paper is to fill in
this gap in the literature and to provide new approximate and exact approaches for TWT-SMSP-PM based on its characteristics and enhanced versions of existing methods in the TWT-SMSP literature. The remaining part of the paper is organised as follows. In section 2, a statement, notation and mixed integer linear formulation for the TWT-SMSP-PM are introduced. Section 3, proposes the new composite heuristic with enhancement of existing dispatching rules. Section 4 presents our extensive computational experience using benchmark instances from the literature and additional maintenance periods with maintenance intervals of different characteristics. Section 5 terminates with a conclusion and further research directions.

2 Problem Statement, Notation and Mixed Integer Linear Formulation

The total weighted tardiness single-machine scheduling problem with periodic maintenance, TWT-SMSP-PM, has a number of assumptions that are stated as follows. On the job side, it is assumed that there exists a set of \( n \) jobs, \( N = \{1, \ldots, n\} \), each job \( i \) has a processing time \( p_i \), a due date \( d_i \), and a penalty \( w_i \) for each unit of time delayed beyond its due date; all \( p_i, d_i, \) and \( w_i \) are integer positive values. It is also assumed that a job is available for processing at time zero without pre-emption, i.e., once the job is started, it must be completed without any interruption. On the machine side, the existence of a set of \( m \) Maintenance Periods \( MP = \{1, \ldots, m\} \) is assumed. Each period is required after a fixed interval of \( T \) units of machine processing time. Each maintenance period is a fixed length of repair time (ML). Here, the objective is to determine a sequence/schedule of jobs on the machine to minimize the total weights of tardy jobs subject to the predetermined schedule of maintenance periods. The following set of notations will be used in the remaining part of the paper:

- \( i \): Index of job \( i \)
- \( p_i \): Processing time of job \( i \) and \( p_i < T \)
- \( d_i \): Due date of job \( i \)
- \( w_i \): Penalty of job \( i \)
- \( c_i \): Completion time of job \( i \)
- \( l_i \): Lateness of job \( i \), where \( l_i = \max(0, c_i - d_i) \)
- \( s_i \): Start of processing time of job \( i \)
- \( n \): Number of jobs
- \( N \): Set of jobs, \( N = \{1, \ldots, n\} \)
- \( P \): The total sum of processing of all jobs, \( P = \sum_{i \in N} p_i \)
- \( AP(j) \): Average processing times of the Pending jobs
- \( \sigma(j) \): Job \( i \) in the \( j \)-th position of the solution sequence
- \( B_k \): Batch \( k \) for processing a subset of jobs within an interval of time \( T \)
- \( T \): Length of processing time for a batch
- \( SL_k \): Slack time of \( B_k \) or the time left after processing jobs in \( B_k \)
- \( M_k \): Maintenance period \( k \)
- \( m \): Number of maintenance periods
- \( m_k \): Starting time of the maintenance period \( k \)
- \( ML \): Length of time to perform the maintenance/repair
- \( M \): Very large positive integer
- \( \alpha \): Look-ahead parameter for the apparent tardiness cost (ATC) dispatching rule
- \( Max_L \): Maximum number of backtrack levels
- \( R \): Due Date Range
- \( T_f \): Tightness Factor
A sequence of jobs and maintenance periods in a given sequence solution $S$ is typically represented by $S = \{B_1, M_1, \ldots, B_k, M_k, \ldots, B_m, M_m, B_{m+1}\}$ where $\sigma(j)$ indicates the order of job $i$ in a batch $B_j$ for $j = 1, \ldots, n$ and $k = 1, \ldots, m + 1$ as depicted in Figure 1.

2.1 Mixed Integer Linear Programming Formulation

This sub-section proposes a mixed integer linear programming (MILP) formulation to the TWT-SMSP-PM. It is inspired from the work of Leon and Wu (1992) and is presented as follows. The objective function of minimizing the total weighted tardiness of jobs can be expressed by a non-linear term as follows: \[
\text{Minimize} \quad NZ = \sum_{i \in N} w_i \times \text{Max}(0, l_i)
\] (1)

Note that the tardiness (lateness) $l_i$ of a job $i$ is either $l_i = c_i - d_i > 0$ or $l_i \leq 0$. Hence, Max $(0, l_i)$ reduces to $l_i$ in (2) with the addition of constraint (3) and the addition of non-negativity constraint $l_i \geq 0$. Consequently, the nonlinear objective (1) can be rewritten as a linear objective function $Z$ as follows:

\[
Z = \text{Minimize} \quad \sum_{i \in N} w_i \times l_i
\] (2)

Subject to:

a) Linearization of the objective function:
\[
c_i = s_i + p_i \quad \text{and} \quad l_i \geq c_i - d_i \quad \forall i \in N
\] (3)

b) Logical relationships among jobs: Either job $i$ starts before job $j$ or $j$ starts before $i$:
\[
c_i \leq s_j \quad \text{or} \quad c_j \leq s_i \quad \forall i, j \in N \quad \text{where} \quad i < j
\] (4)

The logical relationship in equation (4) can be converted into linear forms at the expense of additional set of logical $\delta_{ij}$ variables which are defined as follows. Let

\[
\delta_{ij} = \begin{cases} 
1 & \text{if job } i \text{ precedes job } j \\
0 & \text{otherwise} 
\end{cases} \quad \forall i < j \in N
\]

Consequently, equation (4) can be rewritten using the following two additional constraints expressed in equation (5):
\[
\begin{align*}
&i) \quad c_i \leq s_j + M(1 - \delta_{ij}) \quad \forall i, j \in N \quad i < j \\
&\text{ii)} \quad s_j + M\delta_{ij} \geq c_j \quad \forall i, j \in N \quad i < j
\end{align*}
\] (5)

c) Logical relationships among jobs & maintenance periods: Either job $i$ starts before a maintenance $k$ or $k$ starts before $i$:
\[
c_i \leq m_k \quad \text{or} \quad m_k + ml \leq s_i \quad \forall i \in N, \forall k \in \text{MP}
\] (6)

Similarly, equation (6) can be converted into linear forms as follows:

\[
\gamma_{ik} = \begin{cases} 
1 & \text{if job } i \text{ precedes maintenance } k \\
0 & \text{otherwise} 
\end{cases} \quad \forall i \in N, k \in \text{MP}
\]

\[
\begin{align*}
&i) \quad c_i \leq m_k + M(1 - \gamma_{ik}) \quad \forall i \in N \& \quad k \in \text{MP} \\
&\text{ii)} \quad m_k + ml \leq s_i + M\gamma_{ik} \quad \forall i \in N \& \quad k \in \text{MP}
\end{align*}
\] (7)

d) Setting each maintenance period to start at a constant time:
Set each maintenance period to a known and prefixed constant future time in chronological order. They can be set as follows:
\[
m_k = k \times T + (k - 1) \times ml \quad \forall k \in \text{MP}
\] (8)
3.1 Dispatching Rules

Research in dispatching rules (DR) has been active for several decades and many different rules have been studied in the literature. DR is a priority function to select the next job for processing on the machine. It is based on attributes of a job such as weight, processing time, and due date. Such priority function may be classified as dynamic or static depending on whether the attributes are constant or time-dependent. In a time-dependent case, a job may have different priorities for processing on the machine at different points in time. Table 1 summarises the most well-known dispatching rules in the TWT-SMSP literature. Most of them are based on Emmons [9]’s dominance conditions and their enhancements for TWT-SMSP. Emmons’ dominance properties established precedence relations among jobs in an optimal schedule. For instance, it is well known that the Earliest Due Date rule (EDD) gives an optimal sequence for the TWT-SMSP when all due dates are sufficiently loose or spread out, whereas the WSPT rule gives an optimal solution when either all due dates are zeros or all jobs are tardy. Under this case the TWT-SMSP reduces to the total weighted completion time which is known to be solved optimally by WSPT rule, Smith [21]. WSPT rule sequences jobs in non-increasing order of \( \frac{w_i}{p_i} \). Therefore, it is natural to exploit such nice properties by combining such simple and efficient priority rules in various ways. For instance, Vopsalainen and Morton [24] developed the apparent tardiness cost (ATC) composite rule which processes jobs in non-increasing order of time-dependent index based on a combination of the WSPT rule and the weighted minimum slack (WMS) rule using an exponential function to alternate the DR selection of jobs. Morton and Pentico [17], Volgenant and Teerhuis [23] and Kanet and Li [12] confirmed the superiority of the ATC rule over EDD, WSPT and WRA rules. However, the ATC was comparable to the modified weighted due date rule (WMD) introduced by Kanet and Li [12]. In this section, we are following the same principle of combining the same simple and efficient WSPT and WMS rules, however, using a Max function to alternate job selection. It should be mentioned that all the rules (R1-R6) are from the literature, we refer to the excellent book by Pinedo [18] and the recent paper of Kanet and
Li [12] for more details. However, WMSPT (R7) is our new dispatching rule to be explained in the next section.

Table 1: A set of dispatching-rules for constructing approximate sequences

<table>
<thead>
<tr>
<th>Rule</th>
<th>Definition</th>
<th>Priority Index</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>ATC Apparent Tardiness Cost</td>
<td>$\max \left[ \frac{w_i}{p_i} \exp \left( - \max \left{ \frac{d_i - p_i - t, 0}{\alpha P} \right} \right) \right]$</td>
<td>Dynamic</td>
</tr>
<tr>
<td>R2</td>
<td>WEDD Weighted Earliest Due Date</td>
<td>$\max \left{ \frac{w_i}{d_i} \right}$</td>
<td>Static</td>
</tr>
<tr>
<td>R3</td>
<td>WMDD Weighted Modified Due Date</td>
<td>$\min \left( \max \left( \frac{p_i}{d_i}, \frac{d_i - t}{w_i} \right) \right)$</td>
<td>Dynamic</td>
</tr>
<tr>
<td>R4</td>
<td>WMS Weighted Minimum Slack</td>
<td>$\min \left( \max \left( \frac{d_i - p_i - t, 0}{w_i} \right) \right)$</td>
<td>Dynamic</td>
</tr>
<tr>
<td>R5</td>
<td>WSPT Weighted Shortest Processing Time</td>
<td>$\min \left( \frac{p_i}{w_i} \right)$</td>
<td>Static</td>
</tr>
<tr>
<td>R6</td>
<td>WRA Weighted Remaining Allowance</td>
<td>$\min \left( \frac{d_i - t}{w_i} \right)$</td>
<td>Dynamic</td>
</tr>
<tr>
<td>R7</td>
<td>WMSPT Weighted Minimum Slack Shortest Processing time</td>
<td>$\min \left( \frac{\max(d_i - p_i - t, 0)}{w_i} \right)$</td>
<td>Dynamic</td>
</tr>
</tbody>
</table>

3.2 The Proposed Weighted Minimum Slack & Shortest Processing Time Dispatching Rule (WMSPT) Heuristic

In this section, we first discuss our new proposed composite WMSPT dispatch rule. Second, we introduce a push-back theorem to propose a corrective strategy to fill in slack times created in a batch by any greedy dispatching rule due to the introduction of maintenance constraints. Finally, both the WMSPT rule and the push-back strategy are implemented into what is called a WMSPT heuristic.

3.2.1 WMSPT dispatching rule

Our WMSPT dispatching rule uses the same components of the ATC rule. However, WMSPT is a parameter-free approach which combines the two well-known simple and efficient priority rules, namely, the shortest processing time (WSPT) and the WMS rule. The WSPT rule processes the jobs in non-decreasing order of $\{p_i / w_i\}$ and it is optimal for the TWT-SMSP when all due dates are zeros, i.e. all jobs are tardy, Smith [21]. However, the WMS rule processes the jobs in non-decreasing order of their weighted slack times $\{\max(d_i - p_i - t, 0) / w_i\}$. The WMS rule is also optimal when all due dates are sufficiently spread out, Pinedo [18]. Our new proposed composite rule, which is denoted as WMSPT, attempts to maintain the optimality characteristics of the two dispatching Weighted Minimum Slack and Weighted Shortest Processing Time rules. Under WMSPT, jobs are scheduled one at a time: each time the machine becomes free, a priority ranking index is computed for the remaining unscheduled jobs. The job with the smallest index is then selected for processing next. The ranking index is time dependent and defined as follows:
Clearly WMSPT alternates between WMS and WSPT to capture as much as possible their optimality characteristics. The main advantage of WMSPT lies in its simplicity and the lack of any parameter to control or estimate as compared to ATC.

### 3.2.2 WMSPT heuristic

Generally for scheduling problems, a dispatching rule is normally embedded in what is called a dispatching rule heuristic (DRH). In this section, we shall explain a generic DRH for TWT-SMSP-PM. Initially, a dispatching rule is chosen to generate priority indices for processing unscheduled jobs in an empty schedule consisting of one empty batch available at time 0. The set of unscheduled jobs are then sorted in increasing or decreasing order in UN. DRH then selects the first job in UN and inserts it in the next available position provided it is feasible. A job is feasible for processing in the current processing batch if there is enough slack time. Whenever infeasibility is encountered, a new processing batch is created and the scheduling process is continued until all jobs are scheduled. Note that the only difference between static and dynamic dispatching rules is that in the dynamic case, the priority of selecting a job is updated after each insertion, whereas the priority remains unchanged under the static case. The operation of sorting jobs is therefore executed only once in the static case where it is needed after each update in the dynamic case. The DRH steps are presented as follows:

**Step 1. Initialize:** Given a set of jobs \( N \); select DR; set the following:
- Initial schedule \( S = \phi \); unscheduled jobs \( UN = N \); job position index in the solution sequence \( j = 0 \), and the current time \( t = 0 \).
- the initial batch index \( k = 0 \)

**Step 2. Create new batch:** \( k = k + 1 \); \( B_k = \phi \), \( SL_k = T \) (initial slack equals processing time in a batch)

**Step 3. Rank jobs:** Compute a priority for each unscheduled job in \( UN \) and order jobs in \( UN \) according to the DR ranking (static case only).

**Step 4. Select next job:** Select best job \( *i \) in \( UN \) (first job in the static case).
- If \( p_{*i} \leq SL_k \) then go to Schedule Job (Step 5)
- Else set \( t = m_k + ml \) then go to Create new batch (Step 2).

**Step 5. Schedule Job:** Insert job \( *i \) in batch \( B_k = B_k \cup \{ *i \} \); set \( j = j + 1 \) and insert job \( *i \) immediately after the last position in the current sequence, \( \sigma(j) = *i \); update the processing time \( t = t + p_{*i} \), and the slack time \( SL_k = SL_k - p_{*i} \)

**Step 6. Update list by removing job \( *i \) from \( UN = UN \setminus \{ *i \} \), if \( UN \neq \phi \) then go to Select next job (Step 4) in static case or Rank jobs (Step 3) in dynamic case.

**Step 7. Terminate:** Compute the total weighted tardiness associated with the sequence obtained in solution \( S \).

Note that the above presented DRH is generic and useful when attempting to find a good schedule for the single machine scheduling without periodic maintenance with single objectives such as makespan, total completion time, or maximum lateness. However, the addition of the periodic maintenance constraint creates an additional complexity which makes DRH less effective. Mainly, if a job does not pass the feasibility test, i.e. fit into the remaining slack of the current batch, a new batch is created. As a result, a large slack time may result at the end of the previous batch. Such slack time may be enough to process a lower priority unscheduled job that can better fit in, instead of attempting to process the higher priority job in a
greedy fashion. Such fashion creates more wasted slack time. Therefore, sorting jobs on the basis of one priority rule may not lead to an acceptable schedule. The following pushback theorem is inspired by the previous observation and is used to devise a new enhancement to modify Step 3 of DRH steps to overcome the above weakness.

**Theorem (Pushback)**

Given a solution \( S = \{B_1, M_1, \cdots, B_t, M_t, \cdots, B_m, M_m, B_{m+1}\} \) of ordered batches, if there exists a job \( j \in B_k \) such that job \( j \) is feasible to be processed in the slack time of an earlier batch \( B_c \) (\( c < k \)) then the new solution after the pushback move would always have a total weighted tardiness smaller than or equal to the original one.

**Proof:**

Let \( WTA_j \) and \( WTB_j \) be the weighted tardiness of job \( j \) after the push-back move into \( B_c \) and its earlier weighted tardiness in \( B_k \), respectively. Let \( C_h \) be the completion time of the last job \( h \) in batch \( B_c \) with an available slack time \( SL_c \), \( CB_j \) the completion time of job \( j \) in batch \( B_k \), and \( CA_j \) the completion time after job \( j \) is processed after job \( h \) in \( B_c \). After execution of the feasible move we get \( B_c = B_c \cup \{j\} \) and \( B_k = B_k \setminus \{j\} \). By definition, we have the following relationships:

1. \( p_j \leq SL_c \) and \( CA_j = C_h + p_j \leq m_h + ml \), where the last term is the starting time of the next batch \( B_{c+1} \).
2. Since \( c < k \) then the starting time and maintenance period (\( c \)) is less than or equal to the starting time of batch \( k \) i.e. \( (m_{k-1} + m_k) \leq CB_j \).

Consequently from I and II, we have \( CA_j < CB_j \). Taking away a positive term from both sides\( (d_j > 0) \), we get \( CA_j - d_j < CB_j - d_j \), then \( \max(CA_j - d_j, 0) < \max(CB_j - d_j, 0) \) i.e. we would always have the new lateness of job \( j \) in the new batch smaller than the lateness of job \( j \) in the original batch regardless of its weight. Hence, the proof is complete.

Based on the pushback theorem, the new WMSPT heuristic can be described to construct in single pass a TWT-SMSP-MA solution as follows. The WMPST heuristic follows exactly the same Steps 1-6 of the generic DR heuristic, except when an infeasible job is encountered, a pushback (PB) strategy is applied. The PB strategy attempts to capture the benefit of bringing back low priority jobs to fill-in available slack times at the end of a current batch i.e., instead of creating a new batch, an attempt is made to select the next unscheduled feasible job that can fit into the remaining slack time. Therefore, the WMSPT with PB strategy requires a modification of only Step 4 into Step 4a as follows:

**Step 3a Push-Back Feasibility Check:** Select job \( i^* \) for scheduling from the top of \( UN \) according to the WMSPT ranking index in equation (10).

1. If \( p_{i^*} \leq SL_k \), then go to scheduling (Step 5),
2. Else attempt to select the next job \( b \) that can fit into the current slack time,
3. If there exists no such job, then set \( t = m_k + ML \) and go to create new batch (step 2),
4. Else go to scheduling job \( b \) in Step 4.

Note that the pushback strategy is inspired from the pushback theorem and it can be embedded into any DR heuristic using DR in Table 1. The letter E will be added to the DRH name to denote that the DRH is applied with pushback for example EWMSPT denotes WMSPT with Pushback.

4 Computational Experience
In this section we report our computational experience with the various developed exact and approximate algorithms. All the algorithms are coded in C# and run on a Pentium 4 3.2 GHz personal computer. In the following subsection, the generation of benchmarking data test instances is explained. Next, the effectiveness in solution quality and efficiency in computation time expressed in CPU in one thousand of a second (ms) of the proposed dispatching heuristics are discussed with respect to the characteristics of the problem instances. Further, the impact of the pushback strategy on the solution quality is investigated.

4.1 Benchmarking Data Test Instances

The data test instances for the TWT-SMSP-PM are generated using the original data test instances that were generated for the TWT-SMSP by Crauwels, Potts, and Van Wassenhove [7] and are available in the OR-library. The characteristics of the original data test can be explained as follows. There are 125 test instances that were generated for each problem size \( n \) of 30, 40, 50 and 100 jobs. There are three information data files one for each size namely i30, i40, i50 and i100. Each file contains the test instance ID, the list of \( n \) jobs’ ID, followed by their processing time \( p_i \), weight \( w_i \), and due date \( d_i \) values. All of the instances were randomly generated as follows: For each job \( i \) \((i=1,\ldots,n)\), an integer processing time \( p_i \) and weight \( w_i \) were generated from the uniform distribution \([1,100]\) and \([1,10]\) respectively. Instance classes of varying hardness were generated by using different uniform distributions for generating the due dates \( d_i \). The due date range factors are set to \( R = \{0.2,0.4,0.6,0.8,1\} \) and the due date tightness factors are set to \( TF = \{0.2,0.4,0.6,0.8,1.0\} \), an integer due date \( d_i \) was randomly generated from the uniform distribution interval \([P \times (1 - TF - R / 2), P \times (1 - TF + R / 2)]\), where \( P = \sum_{i=1}^{n} p_i \). Five instances were generated for each of the 25 pairs of \((R, TF)\) values, yielding 125 instances for each instance of size \( n \).

The values of \( TF \) and \( R \) provide characterizations of the instances. For instance, values of \( TF \) close to 1 indicate that the due dates are tight and values close to zero indicate that the due dates are loose. High values of \( R \) indicate a wide due date range whereas small values indicate a narrow due date range.

Our new data test instances for TWT-SMSP-PM have additional maintenance characteristics in addition to job characteristics in the original data. These maintenance additions are explained as follows. For each data test instance, a total of 8 block-durations (T), and a total 5 different maintenance durations (ML) are considered. The original characteristics and the new maintenance setting would lead to a total of 20,000 data instances used to report our computational experience. Moreover, the values for \( T \) and \( ML \) are generated in relation to the processing time of jobs to have different characteristics similar to that used in closely related literature. For instance, Chen [5] generated instances to minimize the weighted total completion time in relation to the upper value \((b)\) of the processing time interval \([p_i \in [a,b]]\) and set \( T = \theta \times b \) with \( \theta = 1.1, 5 \), and 2 and \( ml = \mu \times b \) with \( \mu = 0.3 \) and 0.6. On the other hand, Raza et al. [20] generated instances to minimize the total number of early and tardy jobs for which they are set \( \theta = 1.5, 2, 2.5, & 3 \) with \( \mu = 0.3, 0.33, 1, \& 1.1 \).

For our data set instances, we have generated different block processing and maintenance repair lengths to investigate wider options than those previously used in the literature. For each of the 125 instances, we have generated forty pair combinations of \((\theta, \mu)\) with the following settings: eight values for \( \theta \) with \( \theta = 1.1, 5, 2, 2.5, 3, 4, 5, \& 6 \) and five values for \( \mu \) with \( \mu = 0.3, 0.4, 0.6, 0.8, \& 1.1 \).

Table 2 summarizes the various characteristics of the data instances. The test instances can be characterized with respect to \( \theta \) and \( \mu \) values; small values of \( \theta \) indicate shorter processing time interval \( T \) (one week) where only few jobs can be processed in a batch, whereas medium to large values of \( \theta \) indicate larger processing time interval (one month, or
one quarter) where more jobs can be processed. Small values of $\mu$ indicate shorter repair length (one day) whereas larger $\mu$ values indicate larger repair time (2 days or more).

Table 2  Characteristics of New Data Test Instances

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Variations</th>
<th>Experimented Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jobs characteristics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of jobs</td>
<td>4</td>
<td>30, 40, 50, 100</td>
</tr>
<tr>
<td>Processing time variability</td>
<td>1</td>
<td>[1, 100]</td>
</tr>
<tr>
<td>Weight variability</td>
<td>1</td>
<td>[1, 10]</td>
</tr>
<tr>
<td>Due date range, $R$</td>
<td>5</td>
<td>0.2, 0.4, 0.6, 0.8, 1.0</td>
</tr>
<tr>
<td>Maintenance characteristics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tardiness factor, $TF$</td>
<td>5</td>
<td>0.2, 0.4, 0.6, 0.8, 1.0</td>
</tr>
<tr>
<td>Block Duration time, $T$</td>
<td>8</td>
<td>100, 150, 200, 250, 300, 400, 500, 600</td>
</tr>
<tr>
<td>Maintenance time, $ML$</td>
<td>5</td>
<td>30, 40, 60, 80, 100</td>
</tr>
<tr>
<td>Total test instances</td>
<td></td>
<td>20000</td>
</tr>
</tbody>
</table>

4.2 Average relative percentage deviation calculation

The effectiveness of an algorithm was measured in terms of the objective function value or the relative percentage deviation (RPD) from the best solution found by all algorithms (including the optimal solutions whenever found). Let $H$ and $B$ be a heuristic solution value and the best-known solution value for a given instance respectively. Then, RPD is computed as $\text{RPD} = \left(\frac{H-B}{B}\right) \times 100$. Unfortunately, this calculation of RPD often led to a division by zero, since for some instances a solution with zero total weighted tardiness was obtained. Therefore, throughout the computational analysis it has been assumed that for any set of instances RPD is calculated assuming that $H$ represents the sum of heuristic values for the target set and $B$ represents the sum of the best-known solution values for the set. For example, for $n=30$, the target set is the set of all instances of $n=30$.

4.3 Impact of the pushback strategy on the performance of dispatching rules

In this section, we demonstrate the high impact of the pushback strategy on improving the solution quality of all dispatching rule heuristics. Table 3 reports the averages of the objective values (OV) overall instances for each size of $n$ for all the seven dispatching rules (R1 to R7) listed in Table 1. We have reported the results for the DR heuristics with and without the pushback strategy. The RPD values in Table 3 report the relative percentage deviation of the values of the compared algorithms. The results for ATC are obtained with a look-ahead parameter $\alpha$ computed using $\alpha = 5.5 - \tau - \rho$ where $\tau = \sum d_i / nP$ and $\rho = (d_{\text{max}} - d_{\text{min}}) / P$ as recommended in Pinedo [18].

From the results in Table 3, a number of remarks can be made. First, the performance of each dispatching rule is clearly improved by implementing the pushback strategy with no overhead in CPU time as shown in the last column CPU. On the contrary, CPU time is improved since job ranking that was performed in every iteration whereas it is done only on feasible jobs in the pushback strategy. The relative percentage improvements of the results of the “without pushback” rule from those “with pushback” reported under the column (%Imp) of Table 3 show the magnitude of impact of this simple rule. The maximum improvements were observed with values of 92, 94, 95 and 96% for $n=30$, 40, 50 and 100 respectively for the WMSPT rule, whereas the corresponding values for WMDD are 1% less and are much more variable for the ATC rule. Such good improvements may be due to the flexible structures of the generated solutions by the WMPST, ATC, and WMDD that allowed free spaces towards the end of blocks unlike the other more greedy dispatching rules based on WSPT per say.

Second, the new proposed WMSPT rule performed best among all the seven DR rules on the average in both strategies. It therefore seems that the WMSPT rule is able to balance better the
penalty trade-offs between the lateness of a job and its processing time than its counterpart rules such as WMDD and ATC. Third, it is supervising to notice the huge improvement of WMSPT over its ancestors WSPT and WMS. WMSPT rule seems to be the best of both rules by alternating them over time. It should be also noted without the pushback strategy the new dispatching rule was ranked first on \( n=30 \) and 40 and second on \( n=50 \) and 100 with insignificant differences. But the later requires a number of parameter settings and higher computational efforts. A final point to mention is that we have used Cplex with a cut-off time to solve the proposed mixed integer formulation of the problem, however, only few cases with special characteristics were solved to optimality, with a huge overall average per set worse than any of the above mentioned dispatching rules. Hence its corresponding results are omitted.

Table 3 Comparison of Dispatching Rules with and without pushback strategy

<table>
<thead>
<tr>
<th>n</th>
<th>DRH</th>
<th>With Pushback</th>
<th>Without Pushback</th>
<th>%Imp</th>
<th>Best</th>
<th>Cplex</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OV</td>
<td>RPD</td>
<td>CPU</td>
<td>OV</td>
<td>RPD</td>
<td>CPU</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>30</td>
<td>WEDD</td>
<td>37750</td>
<td>31.13</td>
<td>0.09</td>
<td>54627</td>
<td>89.76</td>
</tr>
<tr>
<td></td>
<td>WMS</td>
<td>52615</td>
<td>82.77</td>
<td>0.21</td>
<td>79187</td>
<td>175.07</td>
</tr>
<tr>
<td></td>
<td>WSPT</td>
<td>36681</td>
<td>27.42</td>
<td>0.09</td>
<td>41803</td>
<td>45.21</td>
</tr>
<tr>
<td></td>
<td>WRA</td>
<td>64433</td>
<td>123.82</td>
<td>0.19</td>
<td>97341</td>
<td>238.13</td>
</tr>
<tr>
<td></td>
<td>ATC</td>
<td>30715</td>
<td>6.70</td>
<td>0.29</td>
<td>36404</td>
<td>26.45</td>
</tr>
<tr>
<td></td>
<td>WMDD</td>
<td>29405</td>
<td>2.14</td>
<td>0.20</td>
<td>36144</td>
<td>25.55</td>
</tr>
<tr>
<td></td>
<td>WMSPT</td>
<td>29334</td>
<td>1.90</td>
<td>0.20</td>
<td>35895</td>
<td>24.69</td>
</tr>
<tr>
<td>40</td>
<td>WEDD</td>
<td>91958</td>
<td>43.76</td>
<td>0.11</td>
<td>128599</td>
<td>101.04</td>
</tr>
<tr>
<td></td>
<td>WMS</td>
<td>111630</td>
<td>74.51</td>
<td>0.29</td>
<td>163715</td>
<td>155.94</td>
</tr>
<tr>
<td></td>
<td>WSPT</td>
<td>77751</td>
<td>21.55</td>
<td>0.11</td>
<td>88760</td>
<td>38.76</td>
</tr>
<tr>
<td></td>
<td>WRA</td>
<td>141926</td>
<td>121.87</td>
<td>0.27</td>
<td>206714</td>
<td>223.16</td>
</tr>
<tr>
<td></td>
<td>ATC</td>
<td>66476</td>
<td>3.92</td>
<td>0.42</td>
<td>78813</td>
<td>23.21</td>
</tr>
<tr>
<td></td>
<td>WMDD</td>
<td>64802</td>
<td>3.31</td>
<td>0.29</td>
<td>78316</td>
<td>22.43</td>
</tr>
<tr>
<td></td>
<td>WMSPT</td>
<td>64782</td>
<td>1.27</td>
<td>0.29</td>
<td>78207</td>
<td>22.26</td>
</tr>
<tr>
<td>50</td>
<td>WEDD</td>
<td>135718</td>
<td>47.90</td>
<td>0.29</td>
<td>190859</td>
<td>107.99</td>
</tr>
<tr>
<td></td>
<td>WMS</td>
<td>166364</td>
<td>81.30</td>
<td>0.98</td>
<td>248044</td>
<td>170.31</td>
</tr>
<tr>
<td></td>
<td>WSPT</td>
<td>113045</td>
<td>23.19</td>
<td>0.29</td>
<td>130153</td>
<td>41.84</td>
</tr>
<tr>
<td></td>
<td>WRA</td>
<td>214920</td>
<td>134.21</td>
<td>0.87</td>
<td>323809</td>
<td>252.87</td>
</tr>
<tr>
<td></td>
<td>ATC</td>
<td>94752</td>
<td>3.26</td>
<td>1.43</td>
<td>114338</td>
<td>24.60</td>
</tr>
<tr>
<td></td>
<td>WMDD</td>
<td>92976</td>
<td>3.32</td>
<td>0.97</td>
<td>114432</td>
<td>24.70</td>
</tr>
<tr>
<td></td>
<td>WMSPT</td>
<td>92864</td>
<td>1.20</td>
<td>0.97</td>
<td>114484</td>
<td>24.76</td>
</tr>
<tr>
<td>100</td>
<td>WEDD</td>
<td>398333</td>
<td>35.97</td>
<td>0.29</td>
<td>593792</td>
<td>102.70</td>
</tr>
<tr>
<td></td>
<td>WMS</td>
<td>556962</td>
<td>90.12</td>
<td>0.98</td>
<td>862395</td>
<td>194.39</td>
</tr>
<tr>
<td></td>
<td>WSPT</td>
<td>372302</td>
<td>27.09</td>
<td>0.29</td>
<td>441542</td>
<td>50.72</td>
</tr>
<tr>
<td></td>
<td>WRA</td>
<td>723217</td>
<td>146.88</td>
<td>0.87</td>
<td>1117936</td>
<td>281.62</td>
</tr>
<tr>
<td></td>
<td>ATC</td>
<td>298449</td>
<td>1.88</td>
<td>1.43</td>
<td>380673</td>
<td>29.95</td>
</tr>
<tr>
<td></td>
<td>WMDD</td>
<td>296897</td>
<td>1.35</td>
<td>0.97</td>
<td>382371</td>
<td>30.53</td>
</tr>
<tr>
<td></td>
<td>WMSPT</td>
<td>295844</td>
<td>0.99</td>
<td>0.97</td>
<td>381144</td>
<td>30.11</td>
</tr>
</tbody>
</table>

In our further analysis we will only consider the best performing DR heuristics with pushback strategy namely WMSPT, WMDD, and ATC. Table 4 presents a summary of the average improvement of the new proposed method. It can be seen that WMSPT is the best performing algorithm in terms of solution quality at a comparable CPU time to WMDD and much faster than ATC.
Table 4 Comparison of Cplex and the best performing heuristics with pushback strategy

<table>
<thead>
<tr>
<th>RPD</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPD</td>
<td>WEDD</td>
<td>WMS</td>
<td>WSPT</td>
<td>WRA</td>
</tr>
<tr>
<td>30</td>
<td>31.13</td>
<td>82.76</td>
<td>27.42</td>
<td>123.82</td>
</tr>
<tr>
<td>40</td>
<td>43.76</td>
<td>74.51</td>
<td>21.55</td>
<td>121.88</td>
</tr>
<tr>
<td>50</td>
<td>47.90</td>
<td>81.30</td>
<td>23.19</td>
<td>134.21</td>
</tr>
<tr>
<td>100</td>
<td>35.97</td>
<td>90.12</td>
<td>27.09</td>
<td>146.88</td>
</tr>
</tbody>
</table>

Further analyses were conducted in Figure 1 with respect to job characteristics and in figure 2 for machine characteristics. The results in figure 1 clearly shows that when both the due date tightness (TF) factor is narrow (small values) the performance of DR heuristics are weak and they improves as TF becomes large value close to 1. In addition, as the due date range change values from small to large, the same performance partner is repeated but with much worse performance. However, despite all TF and R variations, the ranking order of the DR heuristics is kept the same i.e. WMSPT first, WMDD second, and ATC third.

Figure 1: Heuristics Comparison by R and TF for all n

In Figure 2, we investigated the impact of changing the length or maintenance (ML) and working period (T) of the machine. It can be seen that as T increases, the DRH performance improves by having a lower deviation value. It is interesting to note that as T gets very large (over 400 units), the WMSPT and WMDD performance gap gets very close. However, the difference remains significant in favour of WMSPT rule at 0.95% confidence interval using T-test statistics.
5 Conclusion

The most important finding in this paper is that WMSPT is a very competitive rule to the well-known classical dispatching rules in the literature. The pushback strategy was found to improve all the dispatching rules including the WMSPT heuristic which was to be the most effective and efficient strategy among all tested heuristics for the TWT-SMSP-PM with an equal computational complexity. Another advantage of the WMSPT rule is that it is a very simple to implement, parameter free dispatching rule when compared to the most competitive ATC rule which requires the parameter $\alpha$ as estimate. The current research is continuing to determine the impact of the problem’s characteristics on the new dispatching rules and whether the performance of the new proposed dispatching rule can be extended to the periodic maintenance problems with different objectives.

Acknowledgements The authors would like to acknowledge the American University of Beirut Research Board to support the funding of this project.

References