ELSEVIER

European Journal of Operational Research 135 (2001) 394–412

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Theory and Methodology

A new dominance rule to minimize total weighted tardiness with unequal release dates

M. Selim Akturk *, Deniz Ozdemir

Department of Industrial Engineering, Bilkent University, 06533 Bilkent, Ankara, Turkey

Received 26 May 1998; accepted 7 November 2000

Abstract

We present a new dominance rule by considering the time-dependent orderings between each pair of jobs for the single machine total weighted tardiness problem with release dates. The proposed dominance rule provides a sufficient condition for local optimality. Therefore, if any sequence violates the dominance rule then switching the violating jobs either lowers the total weighted tardiness or leaves it unchanged. We introduce an algorithm based on the dominance rule, which is compared to a number of competing heuristics for a set of randomly generated problems. Our computational results indicate that the proposed algorithm dominates the competing algorithms in all runs, therefore it can improve the upper bounding scheme in any enumerative algorithm. The proposed time-dependent local dominance rule is also implemented in two local search algorithms to guide these algorithms to the areas that will most likely contain the good solutions.

Keywords: Scheduling; Heuristics; Single machine; Weighted tardiness; Dominance rule

1. Introduction

We propose a new dominance rule that provides a sufficient condition for local optimality for a single machine total weighted tardiness problem with unequal release dates, $1|r_j|\sum w_jT_j$. Although customer orders may not arrive simultaneously in real-life problems, to the best of our knowledge the authors know of no published exact approach on the $1|r_j|\sum w_jT_j$ problem. The problem may be stated as follows. There are n independent jobs each has an integer processing time p_j, a release date r_j, a due date d_j, and a positive weight w_j. Chu and Portmann [6] shown that the problem can be simplified by using corrected due dates, i.e. if $r_j + p_j > d_j$ then d_j takes the value $r_j + p_j$. Jobs will be processed without interruption on a single machine that can handle only one job at a time. A tardiness penalty is incurred for each time unit

* Corresponding author. Tel.: +90-312-266-4477; fax: +90-312-266-4054.
E-mail address: akturk@bilkent.edu.tr (M.S. Akturk).

0377-2217/01/$ - see front matter
PII: S0377-2217(00)00319-2

if job j is completed after its due date d_j, such that $T_j = \max\{0, (C_j - d_j)\}$, where C_j and T_j are the completion time and the tardiness of job j, respectively. The objective is to find a schedule that minimizes the total weighted tardiness of all jobs given that no job can start processing before its release date. For convenience the jobs are arranged in an EDD indexing convention such that $d_i < d_j$, or $d_i = d_j$ then $p_i < p_j$, or $d_i = d_j$ and $p_i = p_j$ then $w_i > w_j$ or $d_i = d_j$ and $p_i = p_j$ and $w_i = w_j$ then $r_i \leqslant r_j$ for all i and j such that $i < j$.

Rinnooy Kan [20] shows that total tardiness problem with unequal release dates, $1|r_j|\sum T_j$ is NP-hard. Lawler [15] shows that the total weighted tardiness problem, $1||\sum w_jT_j$, is strongly NP-hard, hence unequal release dates problem, $1|r_j|\sum w_jT_j$, is also strongly NP-hard because at least two of its sub problems are already known to be strongly NP-hard. Enumerative solution methods have been proposed for both weighted and unweighted cases when all jobs are initially available. Emmons [9] derives several dominance rules for $1||\sum T_j$ problem that restrict the search for an optimal solution. Rachamadugu [19] and Rinnooy Kan et al. [21] extended these results to $1||\sum w_jT_j$. Szwarc and Liu [24] present a two-stage decomposition mechanism to $1||\sum w_jT_j$ problem when tardiness penalties are proportional to the processing times. Recently, Akturk and Yildirim [1] proposed a new dominance rule and a lower bounding scheme for $1||\sum w_jT_j$ problem that can be used in reducing the number of alternatives in any exact approach.

All the optimizing approaches discussed above assume that the jobs have equal release dates, even though the unequal release dates case has been considered for other optimality criteria. Chu [5] and Dessouky and Deogun [8] give branch and bound (B&B) algorithms to minimize total flow time, $1|r_j|\sum F_j$, whereas Bianco and Ricciardelli [3] and Hariri and Potts [12] consider the total weighted completion time problem, $1|r_j|\sum w_jC_j$. Potts and Van Wassenhove [18] propose a B&B algorithm to minimize the weighted number of late jobs. Erschler et al. [10] establish a dominance relationship within the set of possible sequences for $1 \mid r_j$ problem independent of the optimality criterion to find a restricted set of schedules. Chu [4] proves some dominance properties and provides a lower bound for $1|r_j|\sum T_j$ problem. A B&B algorithm is then constructed using the previous results of Chu and Portmann [6] and problems with up to 30 jobs can be solved for certain problem instances, even though computation requirements for larger problems tend to limit this approach.

2. Dominance rule

The proposed dominance rule provides a sufficient condition for local optimality for the $1|r_j|\sum w_jT_j$ problem, and it generates schedules that cannot be improved by adjacent pairwise job interchanges. If any sequence violates the proposed dominance rule, then switching violating jobs will either lowers the total weighted tardiness or leaves it unchanged. We show that for each pair of jobs, i and j, that are adjacent in an optimal schedule, there can be a critical value t_{ij} such that i precedes j if processing of this pair starts earlier than t_{ij} and j precedes i if processing of this pair starts after t_{ij}. Therefore, the arrangement of two adjacent jobs in an optimal schedule depends on their start time. To introduce the dominance rule, consider schedules $S_1 = Q_1ijQ_2$ and $S_2 = Q_1jiQ_2$ where Q_1 and Q_2 are two disjoint subsequences of the remaining $n-2$ jobs. Let t be the completion time of Q_1. The interchange function $\Delta_{ij}(t)$ gives the cost of interchanging adjacent jobs i and j whose processing starts at time t, and $\Delta_{ij}(t) = f_{ij}(t) - f_{ji}(t)$.

$$f_{ij} = \begin{cases} 0, & \max\{r_i, r_j, t\} \leqslant d_i - (p_i + p_j), \\ w_i(t + p_i + p_j - d_i), & r_j \leqslant t \text{ and } d_i - (p_i + p_j) < t \leqslant d_i - p_i, \\ w_i(r_j + p_j - t), & d_i - p_i \leqslant t < r_j, \\ w_i(r_j + p_i + p_j - d_i), & t \leqslant d_i - p_i \text{ and } t < r_j, \\ w_ip_j, & \max\{r_j, d_i - p_i\} \leqslant t. \end{cases}$$

There are five conditions for the computation of $f_{ij}(t)$. For the first condition, both jobs i and j finish on time, so it is indifferent to schedule either i or j first. In the second condition, job j arrives before time t and job i will become tardy if it is not scheduled first. Therefore, the value of the function $f_{ij}(t)$ is the increasing function of the total weighted tardiness corresponding to the moving job i after job j, i.e. the weighted tardiness of job i. In the third condition, job j arrives strictly after time t and job i will be on time if it is scheduled before job j. Otherwise, job i will be tardy if it is scheduled after job j taking into consideration that there is an idle time on the machine before the beginning of job j, i.e. job i begins at time $r_j + p_j$ instead of t. In the fourth condition, if job i is scheduled before job j, then it can be finished exactly on time, otherwise it will be tardy. Therefore, the value of the function $f_{ij}(t)$ is equal to the weighted tardiness of job i scheduled after job j that begins at time r_j. In the last condition, job j arrives before time t, and job i will be tardy even if it is scheduled before job j. The value of the function $f_{ij}(t)$ is the increasing of the total weighted tardiness corresponding to the moving job i after job j, i.e. job i begins p_j time units later. For the cases 2–5, the formula does not take into account the potential decreasing of the tardiness of job j because it will be considered in $-f_{ji}(t)$.

$\Delta_{ij}(t)$ does not depend on how the jobs are arranged in Q_1 and Q_2 but depends on start time t of the pair since we assume that when the order of the pair of adjacent jobs i and j are inverted this interchange does not delay the beginning of the sequence Q_2, and

- if $\Delta_{ij}(t) < 0$, then j should precede i at time t;
- if $\Delta_{ij}(t) > 0$, then i should precede j at time t;
- if $\Delta_{ij}(t) = 0$, then it is indifferent to schedule i or j first.

It is important to note that the dominance conditions derived for $1||\sum w_jT_j$ problem may not be directly extended to the $1|r_j|\sum w_jT_j$ problem. A global dominance for $1||\sum w_jT_j$ problem implies the existence of an optimal sequence in which job i precedes job j is guaranteed and job i dominates job j for every time point t. An immediate consequence of allowing different release times over the $1||\sum w_jT_j$ problem is the need to examine the question of inserted idle time. To illustrate the role of inserted idle, consider the following three-job example, for which the Gantt charts for three alternative schedules are given in Fig. 1. Let $(\text{Job } j \mid r_j, p_j, d_j, w_j) = (1 \mid 0, 12, 13, 1)$, $(2 \mid 0, 14, 14, 1)$, and $(3 \mid 14, 2, 16, 2)$. If we directly implement dominance rules proposed by Emmons [9], Rinnooy Kan et al. [21], Rachamadugu [19] or Akturk and Yildirim [1], job 1 dominates job 2 for any time $t \geqslant 0$, i.e. global dominance. As shown in Fig. 1(c), the only optimal solution is {2-3-1}, since these rules do not consider the impact of inserted idle time on the final schedule. In Fig. 1(a), the sequence {1-2-3} corresponds to a non-delay schedule, which never permits a delay via inserted idle time when the machine becomes available and there is work waiting.

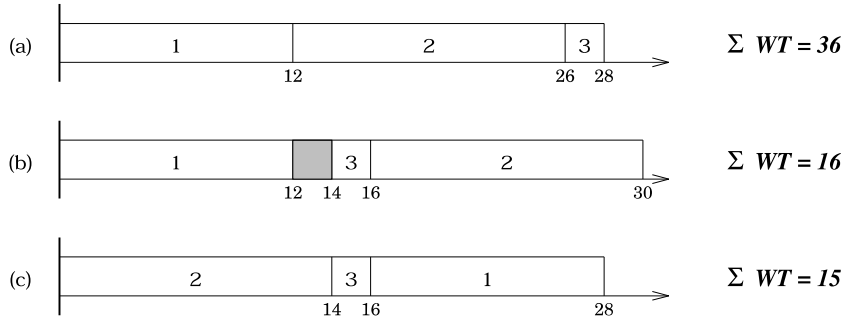

Fig. 1. Three alternative schedules for the three-job example.

The dominance properties for $1|r_j|\sum w_jT_j$ problem can be determined by looking at points where the piecewise linear and continuous functions $f_{ij}(t)$ and $f_{ji}(t)$ intersect. When all of the possible cases are studied, it can be seen that there are at most seven possible intersection points.

$$t_{ij}^1 = [(w_i d_i - w_j d_j)/(w_i - w_j)] - (p_i + p_j), \tag{1}$$

$$t_{ij}^2 = d_j - p_i - p_j(1 - w_i/w_j), \tag{2}$$

$$t_{ij}^3 = d_i - p_j - p_i(1 - w_j/w_i), \tag{3}$$

$$t_{ij}^4 = w_j/w_i(r_i + p_i + p_j - d_j) - (p_i + p_j - d_i), \tag{4}$$

$$t_{ij}^5 = [(w_j - w_i)p_i + w_j r_i + w_i(d_i - p_j)]/(w_i + w_j), \tag{5}$$

$$t_{ij}^6 = w_i/w_j(r_j + p_i + p_j - d_i) - (p_i + p_j - d_j), \tag{6}$$

$$t_{ij}^7 = [(w_i - w_j)p_j + w_i r_j + w_j(d_j - p_i)]/(w_i + w_j). \tag{7}$$

The intersection points are denoted as a breakpoint if they are in their specified intervals as shown below. A *breakpoint* is a critical start time for each pair of adjacent jobs after which the ordering changes direction such that if $t \leqslant$ breakpoint, i precedes j (or j precedes i) and then j precedes i (or i precedes j). When an intersection point precedes the arrival of one of the two considered jobs, then the release date of this job is considered as a breakpoint. At intersection points t_{ij}^4 and t_{ij}^5, job i should precede job j, but job i becomes available after the intersection point, hence r_i is denoted as a breakpoint. Similarly, r_j is denoted as a breakpoint instead of t_{ij}^6 and t_{ij}^7. As a result, interchanging two adjacent jobs using the proposed local dominance rule will not delay the earliest scheduling date for the sequence Q_2 as indicated above.

- t_{ij}^1 will be a breakpoint if $\max\{d_j - (p_i + p_j), r_i, r_j\} < t_{ij}^1 \leqslant \min\{d_i - p_i, d_j - p_j\}$,
- t_{ij}^2 will be a breakpoint if $\max\{d_i - p_i, d_j - (p_i + p_j), r_j\} < t_{ij}^2 < d_j - p_j$,
- t_{ij}^3 will be a breakpoint if $\max\{d_j - p_j, r_i\} \leqslant t_{ij}^3 < d_i - p_i$,
- r_i will be a breakpoint if either $d_i - (p_i + p_j) < t_{ij}^4 \leqslant \min\{d_j - p_j, r_i\}$ or $d_j - p_j < t_{ij}^5 \leqslant r_i$,
- r_j will be a breakpoint if either $d_j - (p_i + p_j) < t_{ij}^6 \leqslant \min\{d_i - p_i, r_j\}$ or $d_i - p_i < t_{ij}^7 \leqslant r_j$.

Throughout the paper, we also use the following definitions. i *conditionally* precedes j, $(i \prec j)$ if there is at least one breakpoint between the pair of jobs such that the order of jobs depends on the start time of this pair and changes in two sides of that breakpoint. i *unconditionally* precedes j, $(i \to j)$ the ordering does not change, i.e. i always precedes j when they are adjacent, but this does not imply that an optimal sequence exists in which i precedes j.

In order to derive a new dominance rule, we analyze 31 exhaustive cases. Detailed proofs of these cases are not included here due to space limitations but can be obtained from the first author [2]. To clarify the background behind the general rule, the following three different cases are investigated as an example. In the first case as it can be seen in Fig. 2 there is a single intersection point, t_{ij}^6. Furthermore, $f_{ij}(t) > f_{ji}(t)$ for $t < t_{ij}^6$, and $f_{ji}(t) > f_{ij}(t)$ afterwards. But the intersection point occurs before both jobs become available, i.e. $t_{ij}^6 < r_j$, hence r_j becomes a breakpoint as discussed in Proposition 1.

Proposition 1. *If* $p_i w_j > p_j w_i$, $r_i < d_j - (p_i + p_j) < r_j < \min\{d_i - p_i, d_j - p_j\}$ *and* $(w_j - w_i)(r_j + p_i + p_j) \geqslant w_j d_j - w_i d_i$, *then* $i \prec j$ if $t < r_j$ *and* $j \prec i$, *afterwards.*

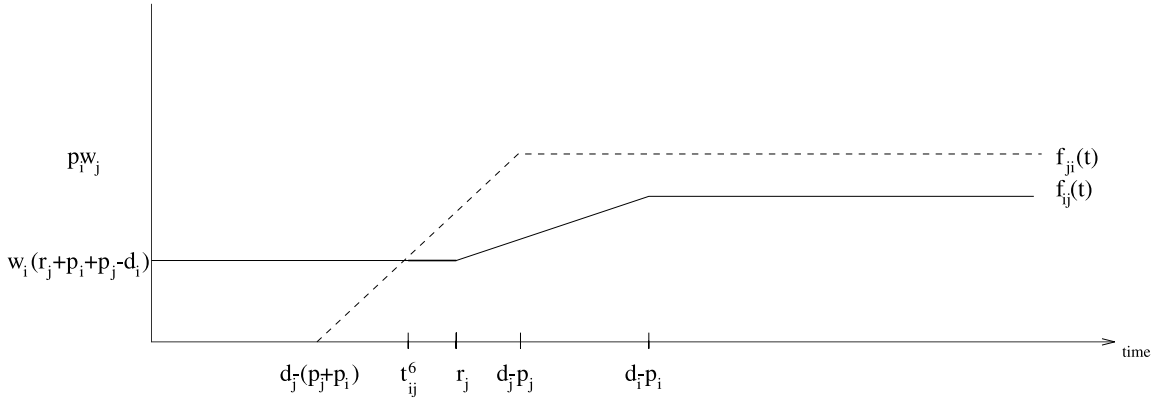

Fig. 2. Illustration of Proposition 1.

Proof. Before r_j, we should schedule job i. As defined earlier $\Delta_{ij}(t) = f_{ij}(t) - f_{ji}(t)$. If we let $t = r_j$ then $\Delta_{ij}(t) = w_i(r_j + p_i + p_j - d_i) - w_j(r_j + p_i + p_j - d_j) \leqslant 0$ since $(w_j - w_i)(r_j + p_i + p_j) \geqslant w_j d_j - w_i d_i$, so $j \prec i$ at $t = r_j$. As $p_i w_j \geqslant p_j w_i$ for $t \geqslant r_j$, $f_{ji}(t) > f_{ij}(t)$ afterwards. Consequently, $\Delta_{ij}(t) < 0$ and $j \prec i$. □

In the second case, there is no breakpoint, which means job j unconditionally precedes job i as shown in Fig. 3. If we can show that $\Delta_{ij}(t) \leqslant 0\ \forall t$, i.e. $f_{ij}(t) \leqslant f_{ji}(t)$ for every t, then $j \to i$ as stated below.

Proposition 2. *If* $r_j \leqslant d_j - (p_i + p_j) \leqslant r_i < d_j - p_j$, $w_j(r_i + p_i + p_j - d_j) \leqslant p_j w_i \leqslant p_i w_j$, *and* $(w_j - w_i)(r_i + p_i + p_j) \geqslant w_j d_j - w_i d_i$, *then* $j \to i$ *for every* t.

Proof. The maximum value of $f_{ij}(t) = p_j w_i$ and the minimum value of $f_{ji}(t) = w_j(r_i + p_i + p_j - d_j)$. If $w_j(r_i + p_i + p_j - d_j) \leqslant p_j w_i \leqslant p_i w_j$, then $f_{ji}(t) \geqslant f_{ij}(t)$ only if $f_{ji}(r_i) \geqslant f_{ij}(r_i)$, i.e. $w_j(r_i + p_i + p_j - d_j) \geqslant w_i(r_i + p_i + p_j - d_i)$. This inequality is equivalent to $(w_j - w_i)(r_i + p_i + p_j) \geqslant w_j d_j - w_i d_i$, so $f_{ji}(t) \geqslant f_{ij}(t)$ for every t leading to $j \to i$. □

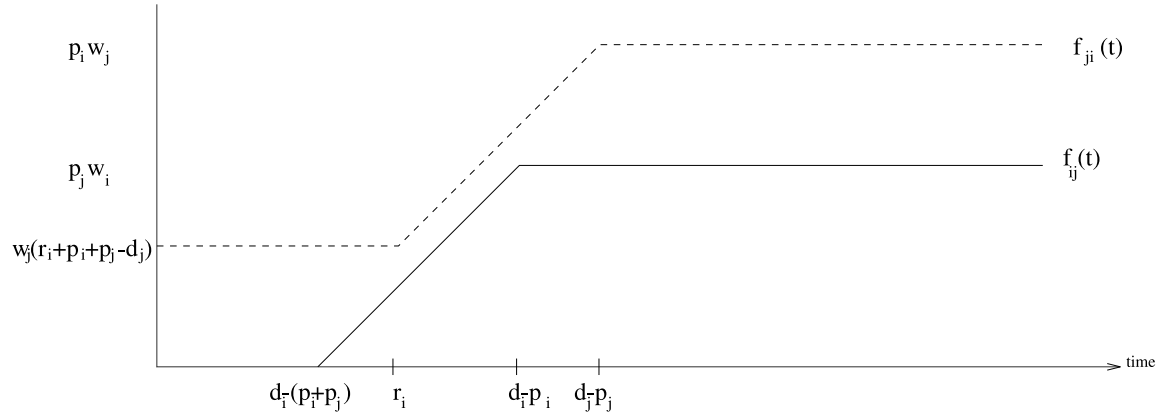

Fig. 3. Illustration of Proposition 2.

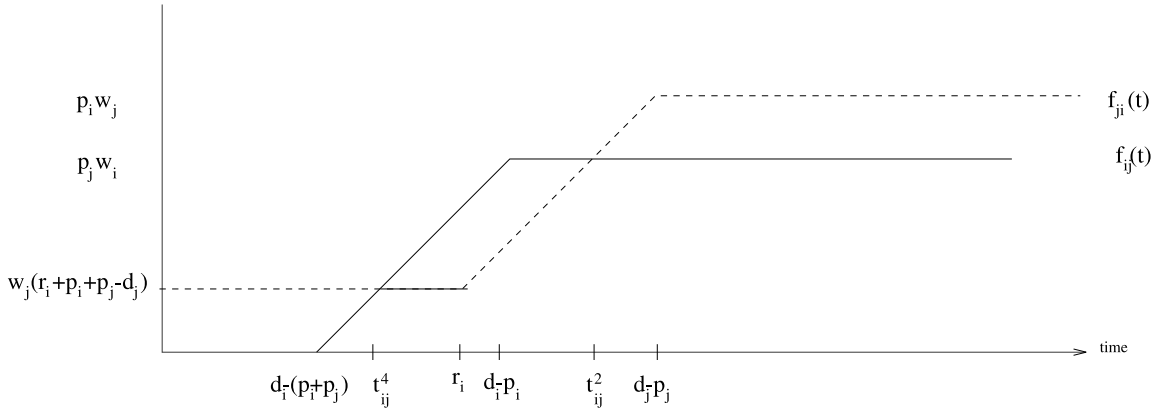

Fig. 4. Illustration of Proposition 3.

The last case is similar to previous one except $d_i - p_i$ is always less than $d_j - p_j$ and the non-constant segment of $f_{ji}(t)$ intersects with the constant segment of $f_{ij}(t)$. As it can be seen from Fig. 4, this difference results in two intersection points t_{ij}^4 and t_{ij}^2. Since $t_{ij}^4 < r_i$, r_i is also denoted as a breakpoint in addition to t_{ij}^2. The following proposition can be used to specify the order of jobs at time t.

Proposition 3. *If* $w_j(r_i + p_i + p_j - d_j) < p_j w_i < p_i w_j$, $p_j(w_i - w_j) > w_j(d_i - d_j)$ *and* $r_j \leqslant d_j - (p_i + p_j) \leqslant r_i \leqslant d_i - p_i < d_j - p_j$ *then there are two breakpoints* r_i *and* t_{ij}^2, *and* $j \prec i$ for $t \leqslant r_i$, $i \prec j$ for $r_i \leqslant t \leqslant t_{ij}^2$ *and* $j \prec i$, *afterwards.*

Proof. Only job j is available until job i arrives at time r_i. After r_i, there is a breakpoint t_{ij}^2 if the non-constant segment of $f_{ji}(t)$ intersects with the constant segment of $f_{ij}(t)$. This is the case if $w_i p_j = w_j(t_{ij}^2 + p_i + p_j - d_j)$ while $d_i - p_i \leqslant t_{ij}^2 < d_j - p_j$. This leads to the condition of $p_i w_j > p_j w_i$ for $t_{ij}^2 < d_j - p_j$ and $p_j(w_j - w_i) \leqslant w_j(d_j - d_i)$ for $t_{ij}^2 \geqslant d_i - p_i$. If $d_j - p_j \leqslant t$ then $j \prec i$ since $\Delta_{ij}(t) = p_j w_i - p_i w_j < 0$. □

After analyzing all possible cases, we show that there are certain time points, called breakpoints, in which the ordering might change for adjacent jobs. As a result, we can state the following general rule, which generates schedules that cannot be strictly improved by one adjacent job interchange.

General Rule

$\text{IF}_{(1)}$ $\max\{d_i - p_i, d_j - (p_i + p_j), r_j\} < t_{ij}^2 < d_j - p_j$
$\text{THEN}_{(1)}$ $\text{IF}_{(2)}$ $r_j < r_i$
  $\text{THEN}_{(2)}$ $j \prec i$ for $t < r_i$,
    $i \prec j$ for $r_i \leqslant t < t_{ij}^2$,
    $j \prec i$ for $t \geqslant t_{ij}^2$,
  $\text{ELSE}_{(2)}$ $i \prec j$ for $t < t_{ij}^2$,
    $j \prec i$ for $t \geqslant t_{ij}^2$,
  $\text{ENDIF}_{(2)}$
$\text{ELSE}_{(1)}$ $\text{IF}_{(3)}$ $\max\{d_j - p_j, r_i\} \leqslant t_{ij}^3 < d_i - p_i$
  $\text{THEN}_{(3)}$ $\text{IF}_{(4)}$ $\max\{d_j - (p_i + p_j), r_i\} < t_{ij}^1 \leqslant d_j - p_j$
    $\text{THEN}_{(4)}$ $\text{IF}_{(5)}$ $r_j < r_i$
      $\text{THEN}_{(5)}$ $j \prec i$ for $t < r_i$,

$i \prec j$ for $r_i \leqslant t \leqslant t_{ij}^1$,
$j \prec i$ for $t_{ij}^1 < t \leqslant t_{ij}^3$,
$i \prec j$ for $t > t_{ij}^3$,
ELSE$_{(5)}$ $i \prec j$ for $t \leqslant t_{ij}^1$,
$j \prec i$ for $t_{ij}^1 < t \leqslant t_{ij}^3$,
$i \prec j$ for $t > t_{ij}^3$,
ENDIF$_{(5)}$
ELSE$_{(4)}$ IF$_{(6)}$ $r_i < r_j$
THEN$_{(6)}$ $i \prec j$ for $t < r_j$,
$j \prec i$ for $r_j \leqslant t \leqslant t_{ij}^3$,
$i \prec j$ for $t > t_{ij}^3$,
ELSE$_{(6)}$ $j \prec i$ for $t \leqslant t_{ij}^3$,
$i \prec j$ for $t > t_{ij}^3$,
ENDIF$_{(6)}$
ENDIF$_{(4)}$
ELSE$_{(3)}$ IF$_{(7)}$ $\max\{r_i, r_j, d_j - (p_i + p_j)\} \leqslant t_{ij}^1 \leqslant \min\{d_i - p_i, d_j - p_j\}$
THEN$_{(7)}$ IF$_{(8)}$ $r_i \leqslant d_j - (p_i + p_j) < r_j$
THEN$_{(8)}$ $i \prec j$ for $t < r_j$,
$j \prec i$ for $r_j \leqslant t < t_{ij}^1$,
$i \prec j$ for $t > t_{ij}^1$,
ELSE$_{(8)}$ IF$_{(9)}$ $r_j < r_i$
THEN$_{(9)}$ $j \prec i$ for $t < r_i$,
$i \prec j$ for $r_i \leqslant t \leqslant t_{ij}^1$,
$j \prec i$ for $t > t_{ij}^1$,
ELSE$_{(9)}$ $i \prec j$ for $t \leqslant t_{ij}^1$,
$j \prec i$ for $t > t_{ij}^1$,
ENDIF$_{(9)}$
ENDIF$_{(8)}$
ELSE$_{(7)}$ IF$_{(10)}$ EITHER $d_j - (p_i + p_j) < t_{ij}^6 \leqslant \min\{d_i - p_i, r_j\}$ OR $d_i - p_i < t_{ij}^7 \leqslant r_j$
THEN$_{(10)}$ $i \prec j$ for $t < r_j$,
$j \prec i$ for $t \geqslant r_j$,
ELSE$_{(10)}$ IF$_{(11)}$ EITHER $d_i - (p_i + p_j) < t_{ij}^4 \leqslant \min\{d_j - p_j, r_i\}$ OR $d_j - p_j < t_{ij}^5 \leqslant r_i$
THEN$_{(11)}$ $j \prec i$ for $t < r_i$,
$i \prec j$ for $t \geqslant r_i$,
ELSE$_{(11)}$ IF$_{(12)}$ $r_i \leqslant r_j$
THEN$_{(12)}$ $i \to j$
ELSE$_{(12)}$ $j \to i$
ENDIF$_{(12)}$
ENDIF$_{(11,10,7,3,1)}$

As we discussed before, there are five breakpoints, namely $t_{ij}^1, t_{ij}^2, t_{ij}^3, r_i,$ and r_j. Let U denote the set of all jobs, V the set of pairs (i, j) for which $\Delta_{ij}(t)$ has at least one breakpoint t_{ij}, $i, j \in V$. We know that both jobs i and j should be available before a breakpoint $t_{ij}^k \geqslant \max\{r_i, r_j\}$ for $k = 1, 2, 3$ so that the largest of these breakpoints is equal to $t_l = \max_{(i,j)\in V}\{t_{ij}^1, t_{ij}^2, t_{ij}^3\}$. The following proposition can be used quite effectively to find an optimal sequence for the remaining jobs on hand after time t_l.

Proposition 4. *If $t > t_l$, then the weighted shortest processing time (WSPT) rule gives an optimal sequence for the remaining unscheduled jobs.*

Proof. t_l is the last breakpoint for any pair of jobs i, j on the time scale. For every job pair (i, j), there is either a breakpoint, t_{ij}, or unconditional ordering $(i \to j)$. The WSPT rule holds for $i \to j$. In the proposed local dominance rule, if both jobs are tardy, that means the current time $t > \max\{t_{ij}^k\}$ for $k = 1, 2, 3$, then the adjacent jobs will be sequenced in non-increasing order of w_j/p_j. If there is a breakpoint then for $t \geqslant t_{ij}$ the job having higher w_j/p_j is scheduled first due to the proposed local dominance rule, so the WSPT order again holds. For $t > t_l$ consider a job i which conflicts with the WSPT rule, then we can have a better schedule by making adjacent job interchanges which either lowers the total weighted tardiness value or leaves it unchanged. If we do the same thing for all of the remaining jobs, we get the WSPT sequence. □

It is a well-known result that the WSPT rule gives an optimal sequence for the $1||\sum w_jT_j$ problem when either all due dates are zero or all jobs are tardy, i.e. $t > \max_{j\in U}\{d_j - p_j\}$. The problem reduces to total weighted completion time problem, $1||\sum w_jC_j$, which is known to be solved optimally by the WSPT rule, in which jobs are sequenced in non-increasing order of w_j/p_j. We know that $t_l \leqslant \max_{j\in U}\{d_j - p_j\}$, so we enlarge the region for which the $1|r_j|\sum w_jT_j$ problem can be solved optimally by the WSPT rule as demonstrated on a set of randomly generated problems in Section 5.

3. Algorithm

Based on the computational complexity results of $1||\sum w_jT_j$ and $1|r_j|\sum T_j$ problems by Lawler [15] and Rinnooy Kan [20], it can be easily deduced that $1|r_j|\sum w_jT_j$ problem is also strongly NP-hard. Since the implicit enumerative algorithms may require considerable computer resources both in terms of computation times and memory, it is important to have a heuristic that provides a reasonably good schedule with reasonable computational effort. Therefore, a number of heuristics have been developed in the literature as summarized in Table 1. MODD, WPD, WSPT, and WDD are examples of static dispatching rules, whereas ATC, COVERT, X-RM, and KZRM are dynamic ones. For the static dispatching rules, the job priorities do not change over time while priorities might change over time for the dynamic dispatching rules. A more detailed discussion on heuristic approaches can be found in Morton and Pentico [16].

Table 1
A set of competing algorithms

Rule	Definition	Rank and priority index
MODD	Earliest modified due date	$\min\{\max\{d_j, t + p_j\}\}$
ATC	Apparent tardiness cost	$\max \pi_j = \left\{\frac{w_j}{p_j}\exp\left(\frac{-\max(0, d_j - t - p_j)}{k\bar{p}}\right)\right\}$
X-RM	X-dispatch ATC	$\max\left\{\pi_j\left(1 - \frac{B\max(0, r_j - t)}{\tilde{p}}\right)\right\}$
COVERT	Weighted cost over time	$\max\left\{\frac{w_j}{p_j}\max\left[0, 1 - \frac{\max(0, d_j - t - p_j)}{kp_j}\right]\right\}$
WPD	Weighted processing due date	$\max\left\{\frac{w_j}{p_jd_j}\right\}$
WSPT	Weighted shortest processing time	$\max\left\{\frac{w_j}{p_j}\right\}$
WDD	Weighted due date	$\max\left\{\frac{w_j}{d_j}\right\}$
KZRM	Kanet and Zhou approach to ATC	ATC with a look-ahead mechanism

The apparent tardiness cost (ATC) is a composite dispatching rule that combines the WSPT rule and the minimum slack rule. Under the ATC rule, jobs are scheduled one at a time; the job with the highest ranking index is then selected to be processed next. The ranking index is a function of the time t at which the machine became free as well as p_j, w_j, and d_j of the remaining jobs. Vepsalainen and Morton [25] have shown that the ATC rule is superior to other sequencing heuristics for the $1||\sum w_jT_j$ problem. It trades off job's urgency (slack) against machine utilization, but due to the more complex weighted criterion, an additional look ahead parameter is needed to assimilate the competing jobs which have different weights. Intuitively, the exponential look ahead works by ensuring timely completion of short jobs (steep increase of priority close to due date), and by extending the look ahead far enough to prevent long tardy jobs from overshadowing clusters of shorter jobs. We set the look-ahead parameter k at 2 as suggested in [16], and $\bar{p}$ is the average processing time of remaining unscheduled jobs at time t.

According to Kanet [13], schedules with inserted idleness appear to have better best case behavior than non-delayed schedules as already shown in Fig. 1. He concluded that non-delay schedules may produce reasonably good performance but rarely provide a schedule which is optimal. Morton and Ramnath [17] modify the ATC rule to allow inserted idleness, which is named the X-RM rule. The X-RM rule can be defined as follows. Whenever a resource is idle, assign it a job which is either available at that time or will be available in the minimum processing time of any job that is currently available. The procedure starts with calculating ATC priorities, $\pi_j(t)$. These priorities are multiplied with $1-[(B\max\{0,r_j-t\})/\tilde{p}]$, hence a priority correction is done to reduce priority of late arriving critical jobs. The parameter B is between 1.6–2, whereas $\tilde{p}$ can be either average processing time, $\bar{p}$, or minimum processing time, $p_{\min}$, as suggested in [16] and [17], respectively. In our study, we compared four different combinations of B and $\tilde{p}$ values such that X-RM I $=(1.6,\bar{p})$, X-RM II $=(2,\bar{p})$, X-RM III $=(1.6,p_{\min})$, and X-RM IV $=(2,p_{\min})$.

The KZRM is a local search heuristic that combines the ATC rule and the decision theory approach of Kanet and Zhou [14]. The decision theory approach defines the alternative courses of action at each decision juncture, evaluate the consequences of each alternative according to a given criterion, and choose the best alternative. In the KZRM rule, we first calculate ATC priorities for all available jobs and generate all possible scenarios putting one of the available jobs first, and ordering the remaining ones by their ATC priorities. After making valuation of each scenario by calculating the objective function, i.e. $F_j=\sum w_jT_j$ where job j is scheduled first, we choose the scenario with the minimum F_j value, and schedule job j. We perform this procedure iteratively until all jobs are scheduled. Therefore, the KZRM rule can be also called a filtered beam search with a beam size of one.

We have proved that the dominance properties provide a sufficient condition for local optimality. Now, we introduce an algorithm based upon the dominance rule that can be used to improve the total weighted tardiness criterion of any sequence S by making necessary interchanges. Let seq$[k]$ denote index of the job in the kth position in the given sequence S and $I[k]$ denote the idle time inserted before kth position in the given sequence S, such that $I[k]=\max\{0,r_{\text{seq}[k]}-C_{\text{seq}[k-1]}\}$. The algorithm can be summarized as follows:

Set $k=1$ and $t=0$.
While $k\leqslant n-1$ do begin
Set $i=\text{seq}[k]$ and $j=\text{seq}[k+1]$
  IF$_{(1)}$ $i<j$ THEN$_{(1)}$
    IF$_{(2)}$ $\max\{d_i-p_i,d_j-(p_i+p_j),r_j\}<t_{ij}^2<d_j-p_j$, and $t_{ij}^2\leqslant t$ THEN$_{(2)}$
      $t=t-p_{\text{seq}[k-1]}-I[k]$, recalculate $I[k]$, change order of i and j, set $k=k-1$
    ELSE$_{(2)}$ IF$_{(3)}$ $\max\{d_j-p_j,r_i\}\leqslant t_{ij}^3<d_i-p_i$ THEN$_{(3)}$
      IF$_{(4)}$ $\max\{d_j-(p_i+p_j),r_i,r_j\}<t_{ij}^1\leqslant\min\{d_i-p_i,d_j-p_j\}$, and $t_{ij}^1<t<t_{ij}^3$ THEN$_{(4)}$
      $t=t-p_{\text{seq}[k-1]}-I[k]$, recalculate $I[k]$, change order of i and j, set $k=k-1$
      ELSE$_{(4)}$ IF$_{(5)}$ $r_j\leqslant t\leqslant t_{ij}^3$ and either $t_{ij}^1\leqslant\max\{d_j-(p_i+p_j),r_i,r_j\}$ or
      $t_{ij}^1>\min\{d_i-p_i,d_j-p_j\}$ THEN$_{(5)}$

$t = t - p_{\mathrm{seq}[k-1]} - I[k]$, recalculate $I[k]$, change order of i and j, set $k = k - 1$
ELSE$_{(5)}$ IF$_{(6)}$ $\max\{d_j - (p_i + p_j), r_i, r_j\} < t_{ij}^1 \leqslant \min\{d_i - p_i, d_j - p_j\}$ THEN$_{(6)}$
IF$_{(7)}$ $r_i \leqslant d_j - (p_i + p_j) < r_j$ and $r_j \leqslant t < t_{ij}^1$ THEN$_{(7)}$
$t = t - p_{\mathrm{seq}[k-1]} - I[k]$, recalculate $I[k]$, change order of i and j, set $k = k - 1$
ELSE$_{(7)}$ IF$_{(8)}$ $t_{ij}^1 < t$ THEN$_{(8)}$
$t = t - p_{\mathrm{seq}[k-1]} - I[k]$, recalculate $I[k]$, change order of i and j, set $k = k - 1$
ELSE$_{(8)}$ IF$_{(9)}$ $t \geqslant r_j$ and either $d_j - (p_i + p_j) < t_{ij}^6 \leqslant \min\{d_i - p_i, r_j\}$ or $d_i - p_i < t_{ij}^7 \leqslant r_j$ THEN$_{(9)}$
$t = t - p_{\mathrm{seq}[k-1]} - I[k]$, recalculate $I[k]$, change order of i and j, set $k = k - 1$
ELSE$_{(9)}$ $t = t + p_i$ and $k = k + 1$.
ENDIF$_{(9)}$
ENDIF$_{(8,7,6,5,4,3,2)}$
ELSE$_{(1)}$ IF$_{(10)}$ $\max\{d_i - p_i, d_j - (p_i + p_j), r_j\} < t_{ij}^2 < d_j - p_j$, and $r_j \leqslant t < t_{ij}^2$ THEN$_{(10)}$
$t = t - p_{\mathrm{seq}[k-1]} - I[k]$, recalculate $I[k]$, change order of i and j, set $k = k - 1$
ELSE$_{(10)}$ IF$_{(11)}$ $\max\{d_j - p_j, r_i\} \leqslant t_{ij}^3 < d_i - p_i$ THEN$_{(11)}$
IF$_{(12)}$ $\max\{d_j - (p_i + p_j), r_i, r_j\} < t_{ij}^1 \leqslant d_j - p_j$, $r_j \leqslant t$ and either $t \leqslant t_{ij}^1$ or $t > t_{ij}^3$ THEN$_{(12)}$
$t = t - p_{\mathrm{seq}[k-1]} - I[k]$, recalculate $I[k]$, change order of i and j, set $k = k - 1$
ELSE$_{(12)}$ IF$_{(13)}$ $t > t_{ij}^3$ and either $t_{ij}^1 \leqslant \max\{d_j - (p_i + p_j), r_i, r_j\}$ or
$t_{ij}^1 > \min\{d_i - p_i, d_j - p_j\}$ THEN$_{(13)}$
$t = t - p_{\mathrm{seq}[k-1]} - I[k]$, recalculate $I[k]$, change order of i and j, set $k = k - 1$
ELSE$_{(13)}$ IF$_{(14)}$ $\max\{d_j - (p_i + p_j), r_i, r_j\} < t_{ij}^1 \leqslant \min\{d_i - p_i, d_j - p_j\}$ THEN$_{(14)}$
IF$_{(15)}$ $r_i \leqslant d_j - (p_i + p_j) < r_j$ and $t > t_{ij}^1$ THEN$_{(15)}$
$t = t - p_{\mathrm{seq}[k-1]} - I[k]$, recalculate $I[k]$, change order of i and j, set $k = k - 1$
ELSE$_{(15)}$ IF$_{(16)}$ $r_i \leqslant t \leqslant t_{ij}^1$ and $r_j \leqslant t$ THEN$_{(16)}$
$t = t - p_{\mathrm{seq}[k-1]} - I[k]$, recalculate $I[k]$, change order of i and j, set $k = k - 1$
ELSE$_{(16)}$ IF$_{(17)}$ $r_j \leqslant t$ and either $d_i - (p_i + p_j) < t_{ij}^4 \leqslant \min\{d_j - p_j, r_i\}$ or $d_j - p_j < t_{ij}^5 \leqslant r_i$ THEN$_{(17)}$
$t = t - p_{\mathrm{seq}[k-1]} - I[k]$, recalculate $I[k]$, change order of i and j, set $k = k - 1$
ELSE$_{(17)}$ $t = t + p_i$ and $k = k + 1$
ENDIF$_{(17)}$
ENDIF$_{(16,15,14,13,12,11,10,1)}$
end.

Let us consider the following 10-job example to explain the proposed algorithm. In this example jobs are initially scheduled by the X-RM II rule. The initial ordering is given in Table 2 along with the sequence, S, release date, r_j, processing time, p_j, weight, w_j, due date, d_j, starting time, t, and weighted tardiness, WT, of each job j. The final schedule after implementing the proposed algorithm on the schedule given by the X-RM II rule is also given in Table 2. The algorithm works as follows: we start from the first job of the given sequence. For each adjacent job pair, we compare the start time of this pair with precedence relations given by the proposed dominance rule. Up to $t = 10$, the sequence generated by the X-RM II rule does not conflict with the dominance rule. But job 7 in the 4th position violates the dominance rule when compared to job 5 in the 5th position at time $t = 10$. The breakpoint $t_{5,7}^2$ is equal to 18.11, which is greater than $t = 10$, that means $5 \prec 7$ at time $t = 10$, so an interchange should be made. There is no idle time before job 7 so $I[3] = 0$, then t is set to $10 - p_{\mathrm{seq}[3]} = 6$ and $k = k - 1 = 3$. Since the job in 4th position is changed, algorithm returns one step back to check the dominance rule between the jobs at position k and $k + 1$, i.e. jobs 2 and 5. We proceed on, another interchange is made at $t = 23$ between jobs 9 and 6, then between jobs 8 and 6, and finally between jobs 10 and 4. Notice that, after all necessary interchanges are performed on the sequence generated by the X-RM II rule, the total weighted tardiness dropped from 61 to 21 giving an improvement of $(61 - 21)/61 = 66\%$. For this example, the optimum solution is also equal to 21.

Table 2
A numerical example

S	X-RM II rule							Dominance rule		
	Jobs	r_j	p_j	w_j	d_j	t	WT	Jobs	t	WT
1	1	1	2	3	10	1	0	1	1	0
2	3	5	1	6	15	5	0	3	5	0
3	2	6	4	4	11	6	0	2	6	0
4	7	9	5	9	28	10	0	5	10	0
5	5	7	6	2	24	15	0	7	16	0
6	8	21	2	7	30	21	0	6	21	5
7	9	21	4	8	36	23	0	8	28	0
8	6	18	7	5	27	27	35	9	30	0
9	10	18	10	9	49	34	0	4	34	16
10	4	11	5	1	23	44	26	10	39	0
Total weighted tardiness							61			21

4. Computational results

We tested the proposed algorithm on a set of randomly generated problems on a Sun-Sparc 1000E server using Sun Pascal. The proposed algorithm was compared with a number of heuristics on problems with 50, 100, and 150 jobs that were generated as follows. For each job j, an integer processing time p_j and an integer weight w_j were generated from two uniform distributions [1, 10] and [1, 100] to create low or high variation, respectively. Instead of finding due dates directly, we generated slack times between due dates and earliest completion times, i.e. $d_j - (r_j + p_j)$, from a uniform distribution between 0 and $\beta \sum_{j=1}^{n} p_j$, whereas release dates, r_j, are generated from a uniform distribution ranging from 0 to $\alpha \sum_{j=1}^{n} p_j$ as in Chu [4]. As summarized in Table 3, a total of 144 example sets were considered and 20 replications were taken for each combination, giving 2880 randomly generated runs.

We have claimed that if any sequence violates the dominance rule, then the proposed algorithm either lowers the weighted tardiness or leaves it unchanged. In order to show the efficiency of the proposed approach, a number of heuristics were implemented on the same problem sets. The proposed algorithm starts from the first job of the given sequence and proceed on as outlined in Section 3. The results, which are averaged over 960 runs for each heuristic, are tabulated in Tables 4–6 for 50, 100, and 150 jobs, respectively. For each heuristic, the average weighted tardiness before and after implementing the proposed algorithm along with the average improvement, ($\overline{\text{improv}}$), the average real time in centiseconds used for the heuristic and algorithm, and the average number of interchanges, ($\overline{\text{interch}}$), are summarized. Although the real time depended on the utilization of system when the measurements were taken, it was a good indicator for the computational requirements, since the CPU times were so small that we could not measure them accurately. In general, the actual CPU time is considerably smaller than the real time. Finally, we performed a paired t-test for the difference between the total weighted tardiness values given by the heuristic before and

Table 3
Experimental design

Factors	# of levels	Settings
Number of jobs	3	50, 100, 150
Processing time variability	2	[1, 10], [1, 100]
Weight variability	2	[1, 10], [1, 100]
Release date range, α	4	0.0, 0.5, 1.0, 1.5
Due date range, β	3	0.05, 0.25, 0.5

Table 4
Computational results for $n = 50$

Heuristic	$\sum w_j T_j$		$\overline{\text{Improv}}$ (%)	Real time		$\overline{\text{Interch}}$	t-test value
	Before	After		Before	After		
MODD	147938	143623	3.9	2.25	1.23	7.42	11.47
ATC	98061	96994	7.6	2.52	1.11	12.12	13.71
X-RM I	98646	97359	9.6	4.49	1.04	12.04	12.53
X-RM II	98232	97027	9.3	3.83	1.30	11.98	13.59
X-RM III	98242	96975	10.5	3.99	1.26	12.30	12.66
X-RM IV	97706	96540	9.4	3.94	1.21	12.10	12.76
COVERT	100056	99656	2.0	2.67	0.95	1.68	11.77
WPD	111425	100545	30.8	2.09	2.05	55.17	13.53
WSPT	111480	100319	32.1	1.95	2.08	49.06	11.05
WDD	133086	120018	20.5	1.73	1.66	36.44	10.42
KZRM	96142	96059	0.3	820.10	1.08	0.93	7.70

Table 5
Computational results for $n = 100$

Heuristic	$\sum w_j T_j$		$\overline{\text{Improv}}$ (%)	Real time		$\overline{\text{Interch}}$	t-test value
	Before	After		Before	After		
MODD	626526	612541	2.5	8.19	4.19	22.64	12.89
ATC	410803	408156	6.0	9.94	3.26	29.47	15.29
X-RM I	414423	411303	6.5	15.98	3.72	29.94	14.73
X-RM II	413506	410406	6.7	15.80	3.67	29.51	14.43
X-RM III	412953	410056	6.0	15.62	3.17	29.47	15.28
X-RM IV	412131	409261	5.8	15.31	3.37	29.48	15.39
COVERT	417285	416191	1.3	8.81	3.83	3.79	12.98
WPD	485685	436516	31.3	7.16	5.91	216.76	13.86
WSPT	474567	428902	32.0	6.99	6.10	181.87	10.62
WDD	600836	539149	18.3	7.33	5.54	133.36	10.96
KZRM	405050	404791	0.7	12309.86	3.27	2.43	10.04

Table 6
Computational results for $n = 150$

Heuristic	$\sum w_j T_j$		$\overline{\text{Improv}}$ (%)	Real time		$\overline{\text{Interch}}$	t-test value
	Before	After		Before	After		
MODD	1390614	1366306	1.9	18.54	7.8	42.13	13.23
ATC	909104	904634	4.7	21.91	7.76	48.53	15.06
X-RM I	935756	930374	5.4	37.06	7.72	48.50	12.26
X-RM II	934892	929421	5.6	36.66	7.31	48.16	12.20
X-RM III	914631	909759	6.8	37.43	7.52	48.39	15.00
X-RM IV	913304	908522	5.5	37.39	7.30	47.85	14.94
COVERT	919108	917518	1.1	20.69	7.59	5.64	13.22
WPD	1091802	975884	31.2	14.97	13.15	466.43	13.01
WSPT	1050406	950104	31.4	15.29	12.49	380.16	10.15
WDD	1373981	1250450	20	15.43	10.75	243.79	10.69
KZRM	895869	895406	0.3	65801.26	9.35	4.21	11.32

after applying the proposed algorithm for each run, and these t-test values are reported in the last column. A large t-test value indicates that there is a significant difference between the total weighted tardiness values. The average improvement for each run is found as follows: improv $= \{(F(S^h) - F(S^{DR}))/F(S^h)\} \times 100$,

if $F(S^h) \neq 0$, and zero otherwise, where $F(S^h)$ is the total weighted tardiness value obtained by the heuristic and $F(S^{DR})$ is the total weighted tardiness obtained by the algorithm, which takes the sequence generated by the heuristic as an input.

Among the competing rules, a local search-based KZRM rule performs better than the other rules, although it requires considerably higher computational effort than others. The static MODD, WDD, WPD, and WSPT rules perform poorly in a dynamic environment since they do not consider availability of jobs while sequencing them. Furthermore, quite large t-test values on the average improvement indicate that the proposed algorithm not only dominates the competing rules but also provides a significant improvement on all rules, and the amount of improvement is notable at 99% confidence level for all heuristics. When we analyze the individual heuristics, we perform 12.1 pairwise interchanges on the average for the X-RM IV rule and improve the results by 9.4% for 50 jobs. On the other hand, the average number of interchanges increases to 55.17 for the WPD rule with a 30.8% improvement. The amount of improvement over the KZRM rule might seem a small percentage, but considering the fact that KZRM rule requires 65801.26 centiseconds on the average to find a schedule for 150 jobs, whereas our proposed algorithm can still improve it by 0.3% after spending only 9.35 centiseconds, which is 1.4×10^{-4} times less than the time required to find an initial schedule. Moreover, we discuss the effect of the range of processing times and weights on the two best rules of X-RM IV and KZRM for 100 jobs case as an example in Table 7. These results are averaged over 240 randomly generated runs, and they indicate the robustness of the proposed algorithm to changing conditions of the experimental factors.

We already showed how the proposed local dominance rule can be used to improve a sequence given by a dispatching rule. The obvious disadvantage of dispatching rules is that the solutions generated by these methods may be far from the optimum. This problem can be tackled by local search methods. Crauwels et al. [7] present several local search heuristics for the $1||\sum w_jT_j$ problem. They introduce a new binary encoding scheme to represent solutions, together with a heuristic to decode the binary representations into actual sequences. This binary encoding scheme is also compared to the usual permutation representation for descent, simulated annealing, threshold accepting, tabu search and genetic algorithms on a large set of problems. We now demonstrate how the proposed dominance rule can be implemented in a local search algorithm, namely on the greedy randomized adaptive search procedure (GRASP) by Feo and Resende [11] and the problem space genetic algorithm (PSGA) by Storer et al. [22].

The GRASP is an iterative process that provides a solution to the problem at the end of each iteration and the final solution is the best one that is obtained during the search as discussed in [11]. GRASP consists of two phases: in the construction phase GRASP builds a feasible schedule iteratively with respect to a greedy function by constructing a restricted candidate list (RCL) and select one job from this list randomly. $\mathrm{RCL} = \{j : \alpha_j \geqslant \alpha\}$ where α_j is the ratio of greedy function score of job j to the highest score obtained at

Table 7
Detailed computational results for $n = 100$

Heuristic		$\sum w_jT_j$		$\overline{\text{Improv}}$ (%)
		Before	After	
X-RM IV	w_{low}, p_{low}	17708	17593	5.1
	w_{low}, p_{high}	156515	155473	5.1
	w_{high}, p_{low}	149672	148689	6.8
	w_{high}, p_{high}	1324627	1315287	6.3
KZRM	w_{low}, p_{low}	17411	17402	0.4
	w_{low}, p_{high}	153878	153780	1.0
	w_{high}, p_{low}	148689	147065	0.6
	w_{high}, p_{high}	1315287	1300914	0.9

that step, and α is a predetermined ratio parameter. We choose X-RM IV as the greedy function since it is the best dispatching rule that takes into account the inserted idle times due to our computational results. There are two different parameters that should be selected. We must decide the number of jobs that enters to the RCL at each iteration of the construction phase, hence we set $\alpha = 0.5$ or 0.8. The second phase is the iterative improvement procedure that tries to locally optimize the schedule obtained from the construction phase, in which we use our algorithm to guarantee a local optimality. Another parameter to be decided is stopping criterion and we iterate the procedure either 100 or 250 times to construct the "best" schedule. The basic outline of the GRASP algorithm is given below. Let n be the number of jobs and L be the maximum number of iterations.

Step 0 [Initialization] Set $z = 0$.

Step 1 [Phase I] Set $k = 0$.

Step 2 Calculate the X-RM IV ranking index for each job j that can be scheduled at iteration k, denoted as $\pi_j(k)$.

Step 3 [Construct the greedy randomized schedule] Find the job h that has the highest ranking index at iteration k. Set $\mathrm{RCL}(k) = \{j : \pi_j(k)/\pi_h(k) = \alpha_j \geqslant \alpha\}$. Select a job randomly from $\mathrm{RCL}(k)$. If $k < n - 1$ then set $k = k + 1$ and go to Step 2.

Step 4 [Phase II: Local optimization] Calculate the maximum breakpoint, $t_l = \max_{(i,j)\in V}\{t_{ij}^1, t_{ij}^2, t_{ij}^3\}$. Apply the proposed dominance rule to find a local minimum in a forward procedure starting from the first job of the given sequence and proceed on as outlined in Section 3. At any iteration, if $t > t_l$ then order the remaining unscheduled jobs according to the WSPT rule. If the current solution is better than the best solution found until now then update the best solution. Set $z = z + 1$. If $z < L$ then go to Step 1, else stop and report the best solution.

In Table 8, we summarize the number of times the value of a heuristic outperforms others or is one of the best ones before and after implementing the algorithm over 960 runs. Notice that more than one heuristic can have the "best" value for a certain run, if there is a tie. Slight changes can occur in the table when GRASP is iterated 250 times as stated in parentheses. It can be seen that before applying the proposed dominance rule GRASP works better than the X-RM IV rule for 50 jobs, such that X-RM IV outperforms other heuristics 138 times while GRASP ($\alpha = 0.8$, 250 iterations) has the best results for 186 times. But after implementing the dominance rule, X-RM IV gives better results in a significantly less computational time.

Table 8
Number of best results

Heuristic	# OF JOBS = 50		# OF JOBS = 100		# OF JOBS = 150	
	Before	After	Before	After	Before	After
MODD *iter.*# = 100 (*iter.*# = 250)	97	100	98	100	93	99
X-RM I *iter.*# = 100 (*iter.*# = 250)	140	330	156	277	136	252 (251)
X-RM II *iter.*# = 100 (*iter.*# = 250	131	323	155	284	123	238 (237)
X-RM III *iter.*# = 100 (*iter.*# = 250)	147	366	183	293	167	320 (319)
X-RM IV *iter.*# = 100 (*iter.*# = 250)	138	385	159	299	149	288 (287)
ATC *iter.*# = 100 (*iter.*# = 250)	82	269	86	220	73	199 (198)
COVERT *iter.*# = 100 (*iter.*# = 250)	100 (99)	134 (132)	95	123	93	118
WPD *iter.*# = 100 (*iter.*# = 250)	6	102	11	80	22	73
WSPT *iter.*# = 100 (*iter.*# = 250)	5	147	6	101	14	87
WDD *iter.*# = 100 (*iter.*# = 250)	9	41	11	38	9	26
KZRM *iter.*# = 100 (*iter.*# = 250)	324 (323)	457 (456)	224 (223)	546 (542)	233	606
GRASP	$\alpha = 0.5$	$\alpha = 0.8$	$\alpha = 0.5$	$\alpha = 0.8$	$\alpha = 0.5$	$\alpha = 0.8$
iter.# = 100	153	183	103	117	106	112
iter.# = 250	154	186	103	121	105	112

For $n = 150$, the average real time consumed for improving X-RM IV is 7.3 centiseconds while the minimum computation time for GRASP is 9379.67 centiseconds for $\alpha = 0.5$ with 100 iterations. When we compare GRASP with the KZRM rule for $n = 50$, GRASP with $\alpha = 0.5$ used 947.46 centiseconds for 100 iterations and 2371.92 centiseconds for 250 iterations, while the KZRM rule gave 324 best results in 820.1 centiseconds and applying the dominance rule increased the number of best results to 457 in 1 centisecond on the average. In sum, our computational results show that a problem guided heuristic such as X-RM or KZRM supported by our proposed dominance rule to ensure local optimality perform better than a random search-based GRASP algorithm in terms of computational time requirements as well as total weighted tardiness.

PSGA have been used successfully in the past on several scheduling problems by Storer et al. [23]. At the heart of a PSGA is a constructive heuristic which maps a problem instance to a sequence. We again use the X-RM IV augmented with the proposed local dominance rule as the constructive heuristic. A PSGA uses the perturbation vector as the encoding of a solution (or chromosome). Note that a perturbation vector δ may be decoded into a sequence by applying the constructive heuristic. Given any sequence, the objective function (total weighted tardiness) can be calculated. Unlike many applications of genetic algorithms to sequencing problems, standard crossover operators may be applied under this encoding. Once a new generation of perturbation vectors has been created, each element of each vector in the new generation may be mutated. The probability of mutating an element is given by the mutation probability tuning parameter 'MUTPROB'. If selected for mutation, the element is replaced by a newly generated uniform $U(-\theta, \theta)$ random perturbation.

A brief outline of the PSGA with the proposed local dominance rule (LDR) is given below. The maximum number of generations (iterations) and the number of initial population of perturbation vectors, L, are set to 200 and 50, respectively. Clearly more iterations will yield better results, but with diminishing returns. 200 generations seems to balance performance and computation time in a reasonable way. Furthermore, we set MUTPROB $= 0.1$ and $\theta = 0.5$ in all experiments.

Step 0 [Initialization] Set $z = 0$.

Step 1 Randomly generate a perturbation vector δ of size n from the uniform distribution of $U(-\theta, \theta)$. Set $k = 0$.

Step 2 Calculate the X-RM IV ranking index for each eligible job j that can be scheduled at iteration k, denoted as $\pi_j(k)$.

Step 3 The X-RM IV priorities $\pi_j(k)$ are normalized into the interval [0,1] yielding a_j as follows:

Let $\pi_{\min}(k) = \min_j \pi_j(k)$ and $\pi_{\max}(k) = \max_j \pi_j(k)$.

Then $a_j(k) = (\pi_j(k) - \pi_{\min}(k))/(\pi_{\max}(k) - \pi_{\min}(k))$.

Step 4 Perturbations are then added to the normalized priorities, and the job with the highest perturbed normalized priority $a_j(k) + \delta_j$ is scheduled next. Set $k = k + 1$. If $k < n$ then go to Step 2.

Step 5 Apply the proposed local dominance rule as discussed in Section 3. At any iteration, if $t > t_l = \max_{(i,j)\in V}\{t_{ij}^1, t_{ij}^2, t_{ij}^3\}$ then order the remaining unscheduled jobs according to the WSPT rule. Calculate total weighted tardiness, and assign it to the value of the perturbation vector. If the current solution is better than the best solution found until now, then update the best solution. Set $z = z + 1$. If $z < L$, then go to Step 1 to generate a new perturbation vector.

Step 6 For a fixed number of iterations do the following steps and report the best solution.

Step 6.1 Select two perturbation vectors randomly, perform random crossing to generate a new perturbation vector, and apply the mutation probability to each element. Find the index of the perturbation vector with the worst objective function value, i.e. the maximum one, and replace it with the new one.

Step 6.2 Apply the base heuristic, Steps 2–5, with the new perturbation vector.

There are different ways to implement an API method. The most obvious one is a strict descent method (STRICT), in which the only adjacent pairwise interchanges leading to a decrease in the objective function

value are accepted. But, there can be many neutral moves for the $1|r_j|\sum w_jT_j$ problem, especially at the beginning of the sequence. In Table 9, we compare the STRICT method with the proposed LDR to improve the initial sequence given by the X-RM IV rule on each test problem. The proposed PSGA and other algorithms were coded in C language, and run on a Sun Ultra 4000 workstation for the same 144 experimental sets with 10 new replications. The results in Table 9 are averaged over 480 runs for each algorithm for each value of n. In Table 10, we compare these alternative algorithms in more detail for each α and β combination for 100-jobs as an example, where the results are averaged over 40 runs. As it can be seen from these tables, both of the API methods provide a significant improvement over the X-RM IV rule. Furthermore, the LDR-based API method is better than the strict API method as expected.

We also experiment with two different base heuristics for the PSGA, one with LDR and one without. Now, we will discuss the importance of capturing local minima information to guide the local search heuristic. For each run, we compare a straightforward PSGA implementation, denoted as PSGA, that uses the X-RM rule as a base heuristic with another using a local dominance rule-based API search, denoted as PSGA + LDR. The difference between PSGA and PSGA + LDR is quite striking as tabulated in Tables 9 and 10. By defining and searching through a set of local minimums, we were able to improve the solution quality in all measures significantly with a relatively small increase in the CPU time. Both PSGA and PSGA + LDR algorithms were used to solve exactly the same problems, which leads to the conclusion that the PSGA + LDR produces significantly less weighted tardiness than the PSGA. It is also important to note that our objective at this stage is neither developing the most efficient local search algorithm for this

Table 9
Comparison of local search algorithms

			X-RM IV	STRICT	LDR	PSGA	PSGA + LDR
$n = 50$	$\sum w_jT_j$	Min	140	0	0	65	0
		Ave	205756	158408	137515	181800	116342
		Max	2148702	2125395	2122368	2148702	1952310
	Improv.	Min	0.0	0.0005	0.0021	0.0	0.0044
		Ave	0.0	0.3891	0.4760	0.1876	0.6141
		Max	0.0	1.0	1.0	0.7341	1.0
	CPU Time	Min	0.0	0.0	0.0	358.0	400.0
		Ave	0.2	1.67	1.66	420.8	471.1
		Max	1.0	5.0	5.0	1124.0	1257.0
$n = 100$	$\sum w_jT_j$	Min	179	0	0	172	0
		Ave	842391	658423	571928	784240	484567
		Max	7666581	7565859	7532090	7666581	7531949
	Improv.	Min	0.0	0.0028	0.0066	0.0	0.0098
		Ave	0.0	0.3599	0.4760	0.1510	0.6220
		Max	0.0	1.0	1.0	0.6614	1.0
	CPU Time	Min	0.0	0.0	0.0	558.0	648.0
		Ave	2.2	2.34	2.52	662.2	766.7
		Max	4.0	5.0	5.0	1052.0	1251.0
$n = 150$	$\sum w_jT_j$	Min	593	0	0	589	0
		Ave	1883721	1523182	1315561	1806107	1148883
		Max	16599755	16502607	16319414	16599755	16319030
	Improv.	Min	0.0	0.0	0.0	0.0	0.013
		Ave	0.0	0.3796	0.4896	0.1178	0.6225
		Max	0.0	1.0	1.0	0.5517	1.0
	CPU Time	Min	2.0	2.0	2.0	1500.0	1702.0
		Ave	5.0	5.26	5.63	1736.8	2000.8
		Max	10.0	10.0	11.0	2927.0	3319.0

Table 10
Results of computational experiments for $n = 100$

α	β		X-RM IV	STRICT		LDR		PSGA		PSGA + LDR	
			$\sum w_jT_j$	$\sum w_jT_j$	Improv.	$\sum w_jT_j$	Improv.	$\sum w_jT_j$	Improv.	$\sum w_jT_j$	Improv.
0.0	0.05	Min	70196	69855	0.0028	69204	0.0067	70196	0.0	42076	0.0098
		Ave	1924361	1898684	0.0116	1890575	0.0169	1924361	0.0	1827851	0.1782
		Max	7666581	7565859	0.0419	7532090	0.0426	7666581	0.0	7531949	0.4658
	0.25	Min	59375	56619	0.0159	45882	0.1351	59375	0.0	43637	0.1655
		Ave	1829220	1662355	0.0694	1437588	0.2060	1791402	0.0152	1348253	0.2540
		Max	6809100	5992967	0.1568	5136124	0.3148	6703911	0.0939	4904726	0.3304
	0.50	Min	55295	48090	0.0174	33127	0.3226	52341	0.0	27883	0.3822
		Ave	1671625	1503197	0.0941	995250	0.4105	1644704	0.0126	885651	0.4677
		Max	6606341	6092684	0.1967	4205683	0.5115	6463177	0.0632	3737524	0.5694
0.5	0.05	Min	34479	25940	0.0267	26489	0.0267	30480	0.0253	24373	0.0862
		Ave	1117840	877235	0.1870	872334	0.1925	969965	0.1265	747104	0.3091
		Max	5011091	3828272	0.3609	3844935	0.3613	3998541	0.2665	3219422	0.4956
	0.25	Min	48329	30858	0.1098	17250	0.0436	46358	0.0323	16395	0.3721
		Ave	1360986	736767	0.4189	800324	0.4202	1212974	0.1028	536136	0.6019
		Max	5525911	3646627	0.6249	4590018	0.7948	5123747	0.2416	2732638	0.8266
	0.50	Min	36209	20665	0.0838	4764	0.2728	35036	0.0	4752	0.5427
		Ave	1219116	695858	0.3829	446456	0.6532	1147394	0.0556	270320	0.7856
		Max	5543098	3367827	0.6235	2753387	0.8924	5301109	0.1561	1470920	0.8926
1.0	0.05	Min	6759	4671	0.072	4766	0.0	4192	0.1012	3082	0.2627
		Ave	190461	146708	0.2159	142531	0.2198	140362	0.2734	95008	0.4853
		Max	803573	647845	0.3788	641729	0.4080	612019	0.4347	456213	0.7096
	0.25	Min	5740	2131	0.1708	1398	0.2089	4531	0.0941	79	0.5745
		Ave	332548	183738	0.4325	154491	0.5570	223220	0.2712	51279	0.8394
		Max	1631781	1048077	0.9054	893493	0.9114	1027453	0.6178	418345	0.9903
	0.50	Min	5534	1341	0.1727	140	0.4773	3351	0.0108	0.0	0.6997
		Ave	300232	145470	0.5260	85800	0.7921	240794	0.2144	34860	0.9175
		Max	1953964	951420	0.9431	887938	0.9913	1677660	0.4697	494235	1.0
1.5	0.05	Min	1039	485	0.1290	401	0.1298	551	0.0833	207	0.3807
		Ave	44879	27376	0.4121	25245	0.4476	30637	0.3037	13334	0.7073
		Max	177717	138124	0.7285	135238	0.7713	132188	0.6614	72486	0.8878
	0.25	Min	781	15	0.3230	15	0.5758	661	0.0415	0.0	0.6827
		Ave	58193	16483	0.7255	6938	0.8891	44778	0.2460	3340	0.9488
		Max	385483	134815	0.9874	61926	0.9912	325981	0.5834	44460	1.0
	0.50	Min	179	0.0	0.1099	0.0	0.3915	172	0.0	0.0	0.7933
		Ave	59225	7205	0.8427	5816	0.8960	40288	0.1907	1673	0.9696
		Max	497524	58339	1.0	49043	1.0	292898	0.6613	16932	1.0

problem nor selecting the best parameter combination for the PSGA. Our PSGA implementation converges rather quickly, hence the initial schedule is very important. We deliberately implemented the proposed dominance rule in a relatively new local search algorithms of GRASP and PSGA, because these algorithms, unlike other local search algorithms, are very sensitive to the quality of the base heuristic.

5. Concluding remarks

In this study, we develop a new algorithm for the $1|r_j|\sum w_jT_j$ problem, which gives a sufficient condition for local optimality. The proposed algorithm is implemented on a set of heuristics including the X-RM and

KZRM rules that are different combinations of ATC rule with the decision theory approach of Kanet and Zhou [14] to implement principles of ATC rule to a dynamic environment. Enumerative algorithms, even for total tardiness problem, require high computational effort. To our knowledge, there is no published exact approach that simultaneously deals with total weighted tardiness problem and unequal release dates. This enhances contribution of our study in the literature. Our computational experiments indicate that the amount of improvement is statistically significant for all heuristics and the proposed algorithm dominates the competing rules in all runs, therefore it can improve the upper bounding scheme in any enumerative algorithm. Furthermore, the time-dependent local dominance rule-based API local search method is a powerful exploitation (intensifying) tool since we know that the global optimum is one of the local optimum solutions. If we search through a set of local optimum solutions, it is most likely that our search space will contain the good solutions as demonstrated on GRASP- and PSGA-based local search algorithms.

Acknowledgements

The authors would like to thank two anonymous referees whose constructive comments have been used to improve this paper.

References

[1] M.S. Akturk, M.B. Yildirim, A new lower bounding scheme for the total weighted tardiness problem, Computers and Operations Research 25 (4) (1998) 265–278.
[2] M.S. Akturk, D. Ozdemir, New dominance properties for $1|r_j|\sum w_jT_j$ problem, Technical Report No. 98-30, Department of Industrial Engineering, Bilkent University, Turkey, 1998.
[3] L. Bianco, S. Ricciardelli, Scheduling of a single machine to minimize total weighted completion time subject to release dates, Naval Research Logistics 29 (1) (1982) 151–167.
[4] C. Chu, A branch-and-bound algorithm to minimize total tardiness with unequal release dates, Naval Research Logistics 39 (1992) 265–283.
[5] C. Chu, A branch-and-bound algorithm to minimize total flow time with unequal release dates, Naval Research Logistics 39 (1992) 859–875.
[6] C. Chu, M.C. Portmann, Some new efficient methods to solve the $n|1 \mid r_i|\sum T_i$ scheduling problem, European Journal of Operational Research 58 (1992) 404–413.
[7] H.A.J. Crauwels, C.N. Potts, L.N. Van Wassenhove, Local search heuristics for the single machine total weighted tardiness scheduling problem, INFORMS Journal on Computing 10 (3) (1998) 341–350.
[8] M.I. Dessouky, J.S. Deogun, Sequencing jobs with unequal ready times to minimize mean flow time, SIAM Journal of Computing 10 (1981) 192–202.
[9] H. Emmons, One machine sequencing to minimize certain functions of job tardiness, Operations Research 17 (4) (1969) 701–715.
[10] J. Erschler, G. Fontan, C. Merce, F. Roubellat, A new dominance concept in scheduling n jobs on a single machine with ready times and due dates, Operations Research 31 (1983) 114–127.
[11] T.A. Feo, M.G.C. Resende, Greedy randomized adaptive search procedures, Journal of Global Optimization 6 (1995) 109–133.
[12] A.M.A. Hariri, C.N. Potts, An algorithm for single machine sequencing with release dates to minimize total weighted completion time, Discrete Applied Mathematics 5 (1983) 99–109.
[13] J.J. Kanet, Tactically delayed versus non-delay scheduling: An experiment investigation, European Journal of Operational Research 24 (1986) 99–105.
[14] J.J. Kanet, Z. Zhou, A decision theory approach to priority dispatching for job shop scheduling, Production and Operations Management 2 (1) (1993) 2–14.
[15] E.L. Lawler, A 'pseudopolynomial' algorithm for sequencing jobs to minimize total tardiness, Annals of Discrete Mathematics 1 (1977) 331–342.
[16] T.E. Morton, D.W. Pentico, Heuristic Scheduling Systems with Applications to Production Systems and Project Management, Wiley, New York, 1993.
[17] T.E. Morton, P. Ramnath, Guided forward search in tardiness scheduling of large one machine problems, in: D.E. Brown, W.T. Scherer (Eds.), in: Intelligent Scheduling Systems, Kluwer Academic Publishers, Hingham, MA, 1995.

[18] C.N. Potts, L.N. Van Wassenhove, Algorithms for scheduling a single machine to minimize the weighted number of late jobs, Management Science 34 (1988) 843–858.
[19] R.M.V. Rachamadugu, A note on weighted tardiness problem, Operations Research 35 (3) (1987) 450–452.
[20] A.H.G. Rinnooy Kan, Machine Scheduling Problems: Classification, Complexity and Computations, Nijhoff, The Hague, 1976.
[21] A.H.G. Rinnooy Kan, B.J. Lageweg, J.K. Lenstra, Minimizing total costs in one-machine scheduling, Operations Research 23 (1975) 908–927.
[22] R.H. Storer, S.D. Wu, R. Vaccari, New search spaces for sequencing problems with application to job shop scheduling, Management Science 38 (10) (1992) 1495–1509.
[23] R.H. Storer, S.D. Wu, R. Vaccari, Local search in problem and heuristic space for job shop scheduling, ORSA Journal on Computing 7 (4) (1995) 453–467.
[24] W. Szwarc, J.J. Liu, Weighted tardiness single machine scheduling with proportional weights, Management Science 39 (5) (1993) 626–632.
[25] A.P.J. Vepsalainen, T.E. Morton, Priority rules for job shops with weighted tardiness costs, Management Science 33 (1987) 1035–1047.