Comp. Appl. Math. (2018) 37:867–895
https://doi.org/10.1007/s40314-016-0370-4

Exact and heuristic algorithms for minimizing Tardy/Lost penalties on a single-machine scheduling problem

K. Kianfar[1] · **G. Moslehi**[2] · **A. S. Nookabadi**[2]

Received: 4 June 2015 / Revised: 17 May 2016 / Accepted: 22 July 2016 / Published online: 10 August 2016

Abstract This paper addresses minimizing Tardy/Lost penalties with common due dates on a single machine. According to this penalty criterion, if tardiness of a job exceeds a predefined value, the job will be lost and penalized by a fixed value. The problem is formulated as an integer programming model, and a heuristic algorithm is constructed. Then, using the proposed dominance rules and lower bounds, we develop two dynamic programming algorithms as well as a branch and bound. Experimental results show that the heuristic algorithm has an average optimality gap less than 2 % in all problem sizes. Instances up to 250 jobs with low variety of process times are optimally solved and for high process time varieties, the algorithms solved all instances up to 75 jobs.

Keywords Scheduling · Tardy/Lost penalty · Integer programming · Heuristic algorithm · Branch-and-bound algorithm · Dynamic programming

Mathematics Subject Classification 90B35 · 90C11 · 90C39 · 90C57

1 Introduction

In this study, we analyze minimizing total Tardy/Lost penalties on a single machine about two common due dates. Every job i $(1 \leq i \leq n)$ has a processing time, p_i, and two common due dates, namely d_1 and d_2. In the case that a job finishes before the first due date, d_1, no penalty is assigned; if the completion time is between d_1 and d_2, the job will be penalized by a tardiness weight, w_i; and finally, the job will be lost if it is completed after the second due date and a fixed amount of penalty, s_i, will be incurred. We can formulate the Tardy/Lost

Communicated by Ernesto G. Birgin.

✉ K. Kianfar
k.kianfar@eng.ui.ac.ir

[1] Faculty of Engineering, University of Isfahan, Isfahan 81746-73441, Iran

[2] Department of Industrial and Systems Engineering, Isfahan University of Technology, Isfahan 84156-83111, Iran

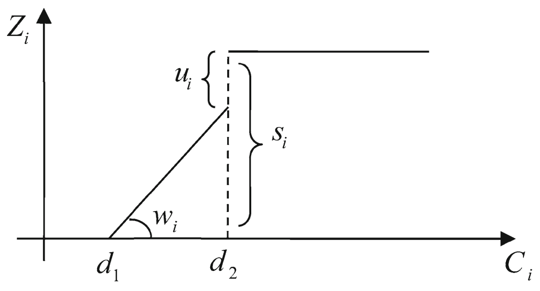

Fig. 1 Tardy/Lost penalty function with common due dates

objective function as in Eq. (1), where C_i is the completion time of job i. It is assumed that $s_i > w_i\,(d_2 - d_1)$ which means penalty for losing a job is greater than the maximum possible tardiness penalty for the same job, and this difference is denoted by parameter u_i.

$$Z_i = \begin{cases} 0 & \text{if } C_i \le d_1 \\ w_i(C_i - d_1) & \text{if } d_1 < C_i \le d_2 \\ s_i = w_i(d_2 - d_1) + u_i & \text{if } C_i > d_2 \end{cases} \tag{1}$$

Figure 1 shows the Tardy/Lost penalty function for job i based on its completion time. The problem is denoted by $1|d_i^1 = d_1, d_i^2 = d_2|\sum_{i\in R_1} w_i T_i + \sum_{i\in R_2} s_i$, where R_1 and R_2 indicate the sets of tardy and lost jobs, respectively. For the sake of brevity, we use $1|d_i^1 = d_1, d_i^2 = d_2|\text{TL}$ in the rest of this paper, where TL indicates the Tardy/Lost performance criteria.

To determine time complexity of this problem, we can set $d_2 \ge \sum_{i=1}^{n} p_i$ and simplify it to the problem of minimizing weighted tardiness with common due date. Yuan (1992) showed that minimizing weighed tardiness with common due date on a single machine is ordinary NP-hard. According to this, the problem considered in this study is at least NP-hard in ordinary sense.

Objective functions of real-life manufacturing problems are often much more complex than the well-known scheduling performance measures. Depending on the type of contractual penalties and expected goodwill of future revenue losses incurred, many types of non-linear tardiness penalty functions may arise. Tardy/Lost measure combines the futures of two parameters; weighted tardiness and weighted number of tardy jobs. On the other hand, the Tardy/Lost performance measure can be considered as a special case for scheduling problems with order acceptance assumption. Here, the objective is minimizing weighted tardiness on a single machine and common due date, where the rejection cost for a job i can be defined as $(d_2 - d_1)w_i$. Order acceptance assumption has been widely investigated in the literature. Engels and Karger (2003) investigated the problem of minimizing weighted completion times and rejection penalties and developed some approximation algorithms. The survey papers by Slotnick (2011) and Shabtay et al. (2013) study a number of scheduling problems with order acceptance.

From a practical point of view, the Tardy/Lost penalty function is applicable in delivery contracts, most of which are arranged based on two due dates. If an order is early, then no penalty is considered; the order will be penalized if its delivery time exceeds the first due date. The penalty increases in proportion to the delivery time until the second due date is reached. If the delivery happens later than the second due date, we will lose the order and a maximum fixed penalty occurs. Other applications of Tardy/Lost measure are perishable goods and food industries.

Table 1 Penalty functions derived from TL measure

Measure	Penalty Function	TL parameter adjustment
Weighted completion time	$Z_i = w_i C_i$	$d_1 = 0$ and $d_2 = \infty$
Weighted tardiness	$Z_i = \begin{cases} 0 & \text{if } C_i \le d_1 \\ w_i (C_i - d_1) & \text{if } C_i > d_1 \end{cases}$	$d_2 = \infty$
Weighted number of tardy jobs	$Z_i = \begin{cases} 0 & \text{if } C_i \le d_1 \\ w_i & \text{if } C_i > d_1 \end{cases}$	$d_1 = d_2$
Late work	$Z_i = \begin{cases} 0 & \text{if } C_i \le d_1 \\ C_i - d_1 & \text{if } d_1 < C_i \le d_2 \\ p_i & \text{if } C_i > d_2 \end{cases}$	$w_i = 1$ and $d_2 = d_1 + p_i$ and $u_i = 0$
Weighted tardiness with order acceptance	$Z_i = \begin{cases} 0 & \text{if } C_i \le d_1 \\ w_i(C_i - d_1) & \text{if } d_1 < C_i \le d_2 \\ R_i & \text{if } C_i > d_2 \end{cases}$	$s_i = R_i =$ *rejection cost* $d_2 =$ *rejection point for tardiness penalty*

Minimizing weighted tardiness is a special case of Tardy/Lost performance measure which has been widely investigated in scheduling area. The problem $1||\sum w_i T_i$ is NP-hard in a strong sense (Lenstra et al. 1977) and is optimally solvable in pseudo-polynomial time for a fixed number of distinct due dates (Kolliopoulos and Steiner 2006). Cheng et al. (2005) presented an $O(n^2)$ time approximation algorithm for the problem as well as a pseudo-polynomial algorithm when all job due dates have equal slacks. Kolliopoulos and Steiner (2006) considered the problem with a fixed number of due dates and designed a pseudo-polynomial algorithm. Karakostas et al. (2009) studied the same problem and designed a dynamic programming algorithm, as well as an approximation scheme. Koulamas (2010) considered the latest theoretical developments for problem $1||\sum T_i$ and reviewed some exact algorithms, fully polynomial time approximation schemes, heuristic algorithms, special cases, and generalizations of the problem.

In a special case of problem $1||\sum w_i T_i$, where due date is common for all jobs, Lawler and Moore (1969) provided a pseudo-polynomial dynamic programming algorithm in $O(n^2 d)$ time and Fathi and Nuttle (1990) developed a 2-approximation algorithm in $O(n^2)$ time. Kellerer and Strusevich (2006) converted a dynamic programming algorithm to an FPTAS of $O(n^6 \log W/\varepsilon^3)$ time complexity, where W is the sum of tardiness weights; later, Kianfar and Moslehi (2013) studied the same problem and developed a more effective FPTAS in $O(n^3/\varepsilon)$ time.

According to Table 1, some of the most common performance measures in scheduling literature are special cases for Tardy/Lost measure. The third column of this table describes how the parameters of TL measure should be adjusted in each case.

Tardy/Lost performance measure is a kind of regular measure which is continuous and non-decreasing in completion times of jobs. Detienne et al. (2012) studied a single-machine problem, whose objective is to minimize a regular step total cost function and they proposed an exact approach based on Lagrangian relaxation. Zhou and Cai (1997) examined two types of regular performance measures, the total cost, and the maximum cost, with general cost functions. In the paper by Shabtay (2008), two continuous and non-decreasing objective functions are considered that include penalties due to earliness, tardiness, number of tardy jobs, and due date assignments. The research by Ventura and Radhakrishnan (2003) was focused on scheduling jobs with varying processing times and distinct due dates on a single machine. Carrasco et al. (2013) studied convex non-decreasing cost functions for single-machine problems subject to precedence constraints. Colin and Quinino (2005) proposed a pseudo-polynomial time algorithm to find a solution within some tolerance of optimality for the same problem. The objective function considered by Baptiste and Pape (2005) was minimizing a regular sum objective function $\sum_i f_i(C_i)$ that corresponds to the cost of the completion of job i at time C_i. They introduced lower bounds and dominance properties for this problem and described a branch-and-bound procedure with constraint propagation. Chandra et al. (2014) developed a binary branch and bound for single-machine problem of minimizing the maximum scheduling cost that is non-decreasing with job completion time.

Pinedo (1995) indicates that, in practice, the penalty function associated with a scheduling problem may follow from a function, in which early jobs are assigned no penalty and those that are finished after their due dates are assigned a penalty that increases at a given rate. Within the penalty function, the job reaches a point, where the penalty assignment changes and increases at a much slower pace. The function identified by Pinedo (1995) is general; however, two more specific functions that react similarly are the deferral cost function (Lawler 1964) and the late work criterion.

Deferral cost functions have been studied by Kahlbacher (1993) who considered general penalty functions monotonous with respect to absolute lateness. He examined several specific cases of the penalty function for situations, in which machine idle times are allowed or not allowed. Federgruen and Mosheiov (1994) considered a class of single-machine scheduling problems with a common due date and general earliness and tardiness penalties. In this study, some polynomial greedy algorithms were proposed and for convex cost structures, they also examined the worst-case ratio bound if the due date is non-restrictive. Baptiste and Sadykov (2009) considered the objective of minimizing a piecewise linear function. They introduced a new Mixed Integer Programming (MIP) model based on time interval decomposition.

The problem of late work minimization on a single machine, as a special case for Tardy/Lost measure, has been addressed in many studies. Potts and Van Wassenhove (1992b) proposed a polynomial time algorithm based on the similarity between tardiness and late work parameters. In another study (Potts and Van Wassenhove 1992a), they developed a branch-and-bound algorithm for the problem which formed a family of approximation algorithms based on truncated enumeration. The results concerning late work are partially reviewed in Chen et al. (1998) and Leung (2004), but Sterna (2011) addresses the first complete review of the topic. Kethley and Alidaee (2002) modified the definition of the late work criterion by introducing two due dates for each job, called due date and deadline. They called the proposed performance criterion as "*modified Total Weighted Late Work*" which is a special case for Tardy/Lost penalty function if we set $u_i = 0$ in Eq. (1).

The main contribution of this research is introducing a new general penalty function, namely Tardy/Lost, for scheduling problems. By adjusting parameters, Tardy/Lost can be converted into some traditional penalty functions, such as weighted tardiness, tardiness with order acceptance assumption, late work, and penalties applied in delivery contracts. From solution point of view, this paper proposes both exact and heuristic methods for a novel problem, in which exact solutions outperform commercial software CPLEX 12 in test instances.

The rest of this paper is organized as follows. In Sects. 2 and 3, we propose a mathematical model and a heuristic algorithm, respectively. Section 4 deals with some dominance rules and lower bounds for problem $1|d_i^1 = d_1, d_i^2 = d_2|\mathrm{TL}$ which later will be used in designing dynamic programming and branch-and-bound algorithms of Sects. 5 and 6. In the next section, we examine the performance of the proposed methods using experimental tests and concluding remarks will be presented in Sect. 8.

2 Mathematical model

To formulate a mixed integer programming model for problem $1|d_i^1 = d_1, d_i^2 = d_2|\mathrm{TL}$, jobs are partitioned into four groups of early, tardy, lost, and straddling. In each optimal solution, all tardy jobs except the first one must be sorted by WSPT (Weighted Shortest Processing Time) ordering. The first tardy job in any optimal schedule may not belong to WSPT, and we call it *straddling job*. The straddling job is started before or at due date d_1 and is finished after d_1. If the straddling job α is finished after d_2, no job belongs to the set of straddling and job α will be considered as lost. However, in the next sections of this paper, it is assumed that there is exactly one straddling job in each schedule and the straddling job may be finished after d_2.

We renumber jobs according to WSPT ordering and define groups of early, tardy, lost, and straddling jobs as groups 1–4, respectively. Decision variables used in this model are as follows.

Z_i is the amount of penalty related to job i.

$X_{i,k}$ is the binary variable that takes value 1 if job i belongs to group k and otherwise is 0.

In the following, we present a mixed integer programming (MIP) model by Eqs. (2) to (10).

$$\text{Min } Z = \sum_{i=1}^{n} Z_i \tag{2}$$

$$\sum_{k=1}^{4} X_{i,k} = 1 \quad \forall i = 1, 2, \ldots, n \tag{3}$$

$$\sum_{i=1}^{n} X_{i,4} \leq 1 \tag{4}$$

$$\sum_{i=1}^{n} \left(p_i . X_{i,1}\right) \leq d_1 \tag{5}$$

$$\sum_{i=1}^{n} \left(p_i . \left(X_{i,1} + X_{i,2} + X_{i,4}\right)\right) \leq d_2 \tag{6}$$

$$Z_i + M_1 \left(1 - X_{i,2}\right) \geq w_i \left[\sum_{j=1}^{n} \left(p_j . \left(X_{j,1} + X_{j,4}\right)\right) + \sum_{j \leq i} \left(p_j . X_{j,2}\right) - d_1\right] \quad \forall i \tag{7}$$

$$Z_i + M_2 \left(1 - X_{i,3}\right) \geq s_i \quad \forall i \tag{8}$$

$$Z_i + M_3 \left(1 - X_{i,4}\right) \geq w_i \left[\sum_{j=1}^{n} \left(p_j . X_{j,1}\right) + p_i - d_1\right] \quad \forall i \tag{9}$$

$$X_{i,k} \in \{0, 1\} \quad \forall i = 1, \ldots, n \quad k = 1, \ldots, 4. \tag{10}$$

Equation (2) indicates the objective function as sum of penalties related to all jobs. Equation (3) assigns each job to one of the previously defined groups. and from Eq. (4). at most one job can be straddling in each schedule. Relation (5) ensures that sum process times of early jobs cannot exceed d_1, while (6) restricts sum process times of early, tardy, and straddling jobs to d_2. Equations (7) to (9), respectively, calculate the penalties related to tardy, lost, and straddling jobs. M_1, M_2, and M_3 are big integers, and supposing, $w_{\max}$, $p_{\max}$, and $s_{\max}$

Table 2 Parameters of jobs in Example 1

Job i	p_i	w_i	s_i
1	2	9	122
2	2	4	44
3	5	8	109
4	7	10	85
5	6	2	18

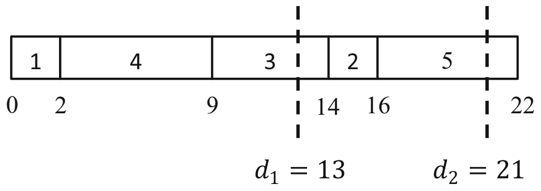

Fig. 2 Optimal sequence for Example 1

represent the maximum values of tardiness weights, process times, and lost penalties, we can define $M_1 = w_{\max}(d_2 - d_1)$, $M_2 = s_{\max}$, and $M_3 = w_{\max} \cdot p_{\max}$.

The above model is original and includes $5n$ variables as well as $4n + 3$ constraints. The number of variables and constraints is significantly small compared with typical models of single-machine scheduling. The following example illustrates the solution of the mathematical model for a small instance.

Example 1 Suppose an instance of problem $1|d_i^1 = d_1, d_i^2 = d_2|\mathrm{TL}$ with 5 jobs. Here, $d_1 = 13$ and $d_2 = 21$, and other parameters are described in Table 2. The optimal sequence $(1, 4, 3, 2, 5)$ will be obtained using the above MIP model, where $x_{1,1} = x_{2,2} = x_{3,4} = x_{4,1} = x_{5,3} = 1$ and other x's are equal to zero. Figure 2 shows completion times of the jobs in the optimal solution. In this solution, jobs 1 and 4 are early, job 3 is straddling, job 2 is tardy, and job 5 is lost. The penalties related to jobs 1–5 are $Z_1 = Z_4 = 0$, $Z_3 = 8$, $Z_2 = 12$, and $Z_5 = 18$ resulting the total penalty $Z = 38$.

3 MTLR heuristic algorithm[1]

In this section, we propose a heuristic algorithm to find near optimal solutions for problem $1|d_i^1 = d_1, d_i^2 = d_2|\mathrm{TL}$. Define the ratio of lost penalty to processing time of a job as penalty ratio. The algorithm tries to schedule lost jobs based on these ratios, and, then, schedules tardy jobs, because lost penalties are greater than tardiness penalties and are considered first. In this procedure, four schedules are built and the best one is selected as output of the algorithm.

This algorithm is composed of two phases, where lost jobs are scheduled in the first phase and tardy jobs are scheduled in the second one. In each iteration of the first phase, a job with minimum s_i/p_i ratio is selected among the set of unscheduled jobs U and will be placed into first idle position from the end of schedule. Moreover, if there exist some jobs able to fill remaining time interval until d_2, the algorithm schedules the one with minimum penalty and saves the obtained secondary schedule, $\hat{G}_1$, besides primary schedule G_1. Considering

[1] Minimum Tardy/Lost Ratio.

$s_i/\min(C_i - d_2, p_i)$ instead of s_i/p_i as ratio for choosing lost jobs creates two other primary and secondary schedules G_2 and $\hat{G}_2$. $\{U_1, U_2, \hat{U}_1,$ and $\hat{U}_2\}$ are the sets of unscheduled jobs in sequences $G_1, G_2, \hat{G}_1$, and $\hat{G}_2$; in addition, $\{j_1, j_2, \hat{j}_1,$ and $\hat{j}_2\}$ are the selected jobs going to be inserted in each iteration of these four schedules.

In the second phase, MTLR selects unscheduled jobs based on minimum $w_i/\min(C_i - d_1, p_i)$ ratios and assigns them as tardy jobs in idle positions from the end of schedules $G_1, \hat{G}_1, G_2$, and $\hat{G}_2$. This makes four complete schedules for the problem, and algorithm MTLR returns the best solution among them as final answer. The time complexity of this algorithm is $O(n^2)$, and its steps are as follows.

1. Set $C_1 = P_{\text{sum}} = \sum_{i=1}^{n} p_i$, $G_1 = \varnothing$, $Z_1 = 0$, $\hat{Z}_1 = \infty$ and $U_1 = \{1, 2, \ldots, n\}$.
2. While $C_1 > d_2$ repeat steps 2.1 and 2.2.
   - 2.1 Select job $j_1 = \underset{i \in U_1}{\operatorname{Arg\,min}} \{s_i/p_i\}$ and schedule it in first free position from the end of G_1. Set $U_1 = U_1|\{j_1\}$, $C_1 = C_1 - p_{j_1}$, $G_1 = (j_1, (G_1))$ and $Z_1 = Z_1 + s_{j_1}$.
   - 2.2 Select job $\hat{j}_1 = \underset{i \in U_1, p_i \geq C_1 - d_2}{Arg\min} \{s_i\}$ if exists and if $Z_1 + s_{\hat{j}_1} < \hat{Z}_1$, then set $\hat{G}_1 = (\hat{j}_1, (G_1))$, $\hat{C}_1 = C_1 - p_{\hat{j}_1}$, $\hat{U}_1 = U_1|\{\hat{j}_1\}$ and $\hat{Z}_1 = Z_1 + s_{\hat{j}_1}$.
3. Repeat steps 1 and 2 letting $j_2 = \underset{i \in U_2}{\operatorname{Arg\,min}} \{s_i/\min(C_i - d_1, p_i)\}$ and create partial schedules G_2 and $\hat{G}_2$ with total penalties Z_2 and $\hat{Z}_2$, respectively.
4. While $C_1 > d_1$ select job $j_1 = \underset{i \in U_1}{\operatorname{Arg\,min}} \{w_i/\min(p_i, C_1 - d_1)\}$. Set $G_1 = (j_1, (G_1))$ $C_1 = C_1 - p_{j_1}$, $U_1 = U_1|\{j_1\}$ and $Z_1 = Z_1 + w_{j_1}(C_1 - d_1)$.
5. Repeat step 4 for schedules $\hat{G}_1, G_2$, and $\hat{G}_2$.
6. Return the best solution among schedules $G_1, \hat{G}_1, G_2$, and $\hat{G}_2$.

4 Dominance rules and lower bounds

In this section, some dominance rules and lower bounds are proposed for problem $1|d_i^1 = d_1, d_i^2 = d_2|$TLwhich later will be used in designing dynamic programming and branch-and-bound algorithms. Suppose that groups of early, tardy, and lost jobs are indicated by σ_E, σ_T, and σ_L.

Dominance rule DR1 Let $i \in \sigma_E$ and $j \in \sigma_T$ be two arbitrary jobs in a sequence satisfying relations $p_i \geq p_j$ and $w_i \leq w_j$. Then, swapping jobs i and j will not increase the total penalty.

Proof The proof is done by comparing total penalty before and after swapping jobs i and j. Figure 3 shows these two cases, where π_1, π_2, and π_3, respectively, denote groups sequenced before, between, and after jobs i and j. Amount of penalty related to groups π_1 and π_3 remains unchanged during swap and completion times of jobs in π_2 will not increase. Let $Z_{\pi_2}^{bef}$ and $Z_{\pi_2}^{aft}$ denote penalty of jobs in group π_2 in these two cases. Therefore, we have

$$\begin{aligned} Z^{\text{bef}} &= Z_{\pi_1} + Z_{\pi_2}^{\text{bef}} + Z_{\pi_3} + w_j(C - d_1) \\ Z^{\text{aft}} &= Z_{\pi_1} + Z_{\pi_2}^{\text{aft}} + Z_{\pi_3} + w_i(C - d_1) \\ &\Rightarrow Z^{\text{aft}} - Z^{\text{bef}} = \left(Z_{\pi_2}^{\text{aft}} - Z_{\pi_2}^{\text{bef}}\right) + \left(w_i - w_j\right)(C - d_1) \leq 0 \end{aligned}$$

and this finalizes the proof. □

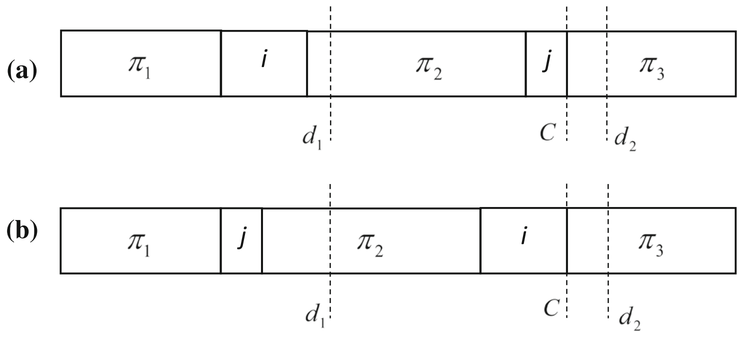

Fig. 3 Sequences related to the proof of DR1

Dominance rule DR2 Let $i \in \sigma_T$ and $j \in \sigma_L$ be two arbitrary jobs in a sequence satisfying relations $p_i \geq p_j$ and $\left(w_i - w_j\right).t + \psi_{i,j} \geq 0$. Then, swapping jobs i and j will not increase the total penalty. Here, t is the start time of job i before swapping and parameter $\psi_{i,j}$ is calculated as $\psi_{i,j} = w_i(p_i - d_1) - w_j(p_j - d_1) + s_j - s_i$.

Proof The proof is done similar to the proof of DR1.

Dominance rule DR3 Let $i \in \sigma_E$ and $j \in \sigma_L$ be two jobs in a sequence satisfying relations $p_i \geq p_j$ and $s_i \leq s_j$. Then, swapping jobs i and j will not increase the total penalty. □

Proof The proof is done similar to the proof of DR1.

Lower bound LB1 This lower bound is composed of two parts; one for lost jobs (LB_L) and the other for tardy jobs (LB_T). Steps of the algorithm for calculating LB_L are as follows.

Algorithm *LB_L*

```
1. Let $U = \{1, 2, \ldots, n\}$ be a set of unscheduled jobs. Set $C = P_{\text{sum}}$ and $\mathrm{LB}_L = 0$.
2. Select a job $i$ from $U$ with minimum $s_i/p_i$ and set $\mathrm{LB}_L = \mathrm{LB}_L + \frac{s_i}{p_i} \min(C - d_1, p_i)$.
3. Set $C = C - p_i$ and $U = U|\{i\}$ and if $C > d_2$ go back to step 2.
```

To calculate LB_T, create an ordered set Ω of artificial jobs via rearranging process times and tardiness weights of the real jobs, such that for each jobs i and $i+1$ in set Ω, we have $p_i \geq p_{i+1}$ and $w_i \leq w_{i+1}$. Jobs are selected from the beginning of set Ω and are scheduled into the tardiness interval from d_2 toward d_1. If last scheduled job passes across d_1, then tardiness penalty is only calculated for part of the job which falls inside the tardiness interval. The following algorithm describes this part of the lower bound.

Algorithm *LB_T*

```
1. Set $C = d_2, i = 1, \mathrm{LB}_T = 0$ and $\Omega = \{(1, 2, \ldots, i, \ldots, n) \mid \forall i p_i \geq p_{i+1}, w_i \leq w_{i+1}\}$.
2. Select i^th artificial job from $\Omega$ and set $\mathrm{LB}_T = \mathrm{LB}_T + \frac{w_i}{p_i}(C - d_1) . \min\{C - d_1, p_i\}$.
3. Set $C = C - p_i, i = i + 1$ and if $C > d_1$, then go back to step 2.
```

Finally, the lower bound will be calculated as $\mathrm{LB1} = \mathrm{LB}_T + \mathrm{LB}_L$. □

Proof Consider minimization of lost penalties as minimizing objects' weights in a knapsack problem, where objects can be split and the knapsack must be filled up. Suppose the knapsack size is $\sum_{i=1}^{n} p_i - d_2$, and in addition, p_i and w_i correspond for size and weight of object i, respectively. Algorithm LB_L selects jobs with minimum penalty ratios, and the ability of splitting jobs guarantee that this algorithm gives a lower bound for the problem of minimizing lost penalties.

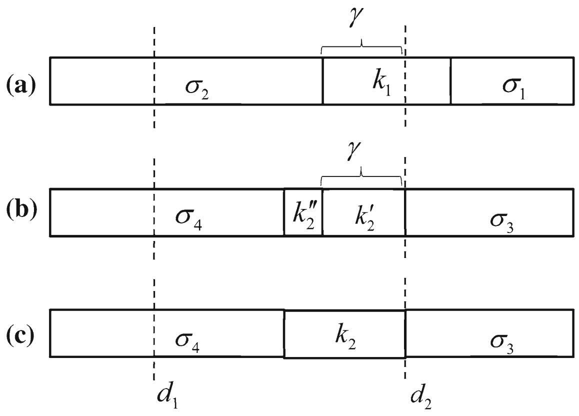

Fig. 4 Sequences used for proving lower bound LB1

Based on the proof of Theorem 2 in reference (Fathi and Nuttle 1990), algorithm LB_*T* gives a lower bound for common due date weighted tardiness problem. The only point is that, algorithm LB_*T* schedules tardy jobs from time d_2, but last tardy job in optimal sequence of problem $1|d_i^1 = d_1, d_i^2 = d_2|\mathrm{TL}$ may finish before d_2. To show that the proposed algorithm gives a lower bound in this case, consider three sequences in Fig. 4.

Let σ_1 to σ_4 be groups of jobs, and according to the definition of artificial jobs in algorithm LB_*T*, job k_2 has the biggest process time, and we split this job into two jobs in the artificial sequence (b) from Fig. 4. Considering $w_{k_2} = w_{k_2'} = w_{k_2''}$, we have $Z_{\sigma_3} \leq Z_{\sigma_1} + s_{k_1}\left(p_{k_1} - \gamma\right)/p_{k_1}$ that implies

$$Z_{\sigma_3} + s_{k_1}\left(\gamma/p_{k_1}\right) \leq Z_{\sigma_1} + s_{k_1}. \tag{11}$$

From $\frac{w_{k_2'}}{p_{k_2}} = \frac{w_{k_2}}{p_{k_2}} \leq \frac{w_i}{p_i} \forall i = 1, \ldots, n$, we will conclude

$$w_{k_2'}\left(d_2 - d_1\right)\left(\gamma/p_{k_2}\right) \leq w_{k_1}\left(d_2 - d_1\right)\left(\gamma/p_{k_1}\right) \leq s_{k_1}\left(\gamma/p_{k_1}\right). \tag{12}$$

By (11) and (12), we get $w_{k_2'}\left(d_2 - d_1\right)\left(\gamma/p_{k_2}\right) + Z_{\sigma_3} \leq Z_{\sigma_1} + s_{k_1}$, and because of $w_{k_2''} \leq w_i \forall i$ we have

$$w_{k_2'}\left(d_2 - d_1\right)\left(\gamma/p_{k_2}\right) + Z_{\sigma_3} + Z_{k_2''} + Z_{\sigma_1} \leq Z_{\sigma_2} + Z_{\sigma_1} + s_{k_1}. \tag{13}$$

Now, we will show $Z_{k_2} \leq Z_{k_2''} + w_{k_2'}\left(d_2 - d_1\right)\left(\gamma/p_{k_2}\right)$ which completes the proof.

$$\begin{aligned} Z_{k_2''} + w_{k_2'}\left(d_2 - d_1\right)(\gamma/p_{k_2}) &= w_{k_2}\left(d_2 - \gamma - d_1\right) + w_{k_2'}\left(d_2 - d_1\right)(\gamma/p_{k_2}) \\ &= w_{k_2}\left(d_2 - d_1\right) + w_{k_2}.\gamma.\left(\frac{d_2 - d_1}{p_{k_2}} - 1\right) \\ &\geq w_{k_2}\left(d_2 - d_1\right) = Z_{k_2}. \end{aligned} \tag{14}$$

□

Lower bound LB2 In this lower bound, the same as LB1, penalties of tardy and lost jobs are calculated separately, and then, we add them up to make the final lower bound. Penalty related to lost jobs is achieved by algorithm LB_*L*, but we use the approximation algorithm from (Fathi and Nuttle 1990) to create a lower bound for tardy jobs. Fathi and Nuttle (1990) propose an approximation algorithm with worst-case ratio bound 2 for problem $1|d_i = d|w_iT_i$. If we divide the penalty from this algorithm by 2, we will get a lower bound for tardy jobs in our problem. In the following, the procedure is described as algorithm LB'_T.

Algorithm LB'_T

```
1. Set $C = d_2, i = 1, LB'_T = 0$, and renumber jobs according to WSPT order.
2. Select job $i$ and calculate $LB'_T = LB'_T + \frac{w_i}{p_i}(C - d_1).\min\{C - d_1, p_i\}$.
3. Set $C = C - p_i, i = i + 1$ and if $C > d_1$, then go back to step 2.
4. Return $LB'_T = LB'_T/2$
```

Therefore, the final lower bound LB2 will be calculated as $\text{LB2} = LB'_T + \text{LB}_L$.

5 Dynamic programming algorithms

In this section, we will propose two dynamic programming algorithms for problem $1|d_i^1 = d_1, d_i^2 = d_2|\text{TL}$. Suppose that jobs are renumbered according to WSPT before developing these DP algorithms. Each optimal schedule for the considered problem includes four parts as: (1) a group of early jobs with any arbitrary order, (2) a straddling job, (3) a group of tardy jobs with WSPT order, and (4) a group of lost jobs with arbitrary order.

5.1 Algorithm DP1

In this dynamic programming algorithm, each state from state space $\nu_k^{(\alpha,C_\alpha)}$ indicates a partial sequence for first k jobs excluding straddling job α that finishes at time C_α. Each state is denoted by a vector $[t_1, t_2, f]$, where t_1 and t_2 show sum of process times for early and tardy groups of jobs, respectively, and f is the penalty for corresponding partial sequence.

Figure 5 shows a state $[t_1, t_2, f]$ from state space $\nu_{k-1}^{(\alpha,C_\alpha)}$ as well as three states derived by adding the upcoming job j to different groups of early, tardy, and lost jobs.

Let $Z^*_{(\alpha,C_\alpha)}$ denote optimal penalty for the problem when job α is straddling with completion time C_α. Steps of the algorithm are as follows.

```
1. For each $\alpha \in \{1, 2, \ldots, n\}$and each $C_\alpha \in [d_1 + 1, d_1 + p_\alpha]$.
    1.1 Set $\nu_0^{(\alpha,C_\alpha)} = \{[0, 0, 0]\}$.
    1.2 For each $k \in \{1, 2, \ldots, \alpha - 1, \alpha + 1, \ldots, n\}$ consider all states $[t_1, t_2, f]$ in $\nu_{k-1}^{(\alpha,C_\alpha)}$.
        • (job $k$ is early) If $t_1 + p_k \le C_\alpha - p_\alpha$ and DR1 and DR3 do not eliminate the new
          state for job $k$, then add state $[t_1 + p_k, t_2, f]$to $\nu_k^{(\alpha,C_\alpha)}$.
        • (job $k$ is tardy) If $C_\alpha + t_2 + p_k \le d_2$ and DR1, DR2, LB1, and LB2 do not eliminate
          the new state for job $k$, then add state $[t_1, t_2 + p_k, f + w_k(C_\alpha + t_2 + p_k)]$to $\nu_k^{(\alpha,C_\alpha)}$.
        • (job $k$ is lost) If $\sum_{i=1}^{k} p_i - t_1 - t_2 < P_{\text{sum}} - d_2$ and DR2, DR3, LB1, and LB2
          do not eliminate the new state for job $k$, then add state $[t_1, t_2, f + s_k]$to $\nu_k^{(\alpha,C_\alpha)}$.
        • For all the states $[t_1, t_2, f] \in \nu_k^{(\alpha,C_\alpha)}$ with equal values for $t_1$ and $t_2$, keep at most
          one state having the minimum value of $f$.
        • Delete the state space $\nu_{k-1}^{(\alpha,C_\alpha)}$.
    1.3 Set $Z^*_{(\alpha,C_\alpha)} = \min_{[t_1,t_2,f]\in\nu_{n-1}^{\alpha,C_\alpha}} \{f\} + Z_\alpha$.
2. Return $Z = \min_{\alpha,C_\alpha} Z^*_{(\alpha,C_\alpha)}$ as optimal solution.
```

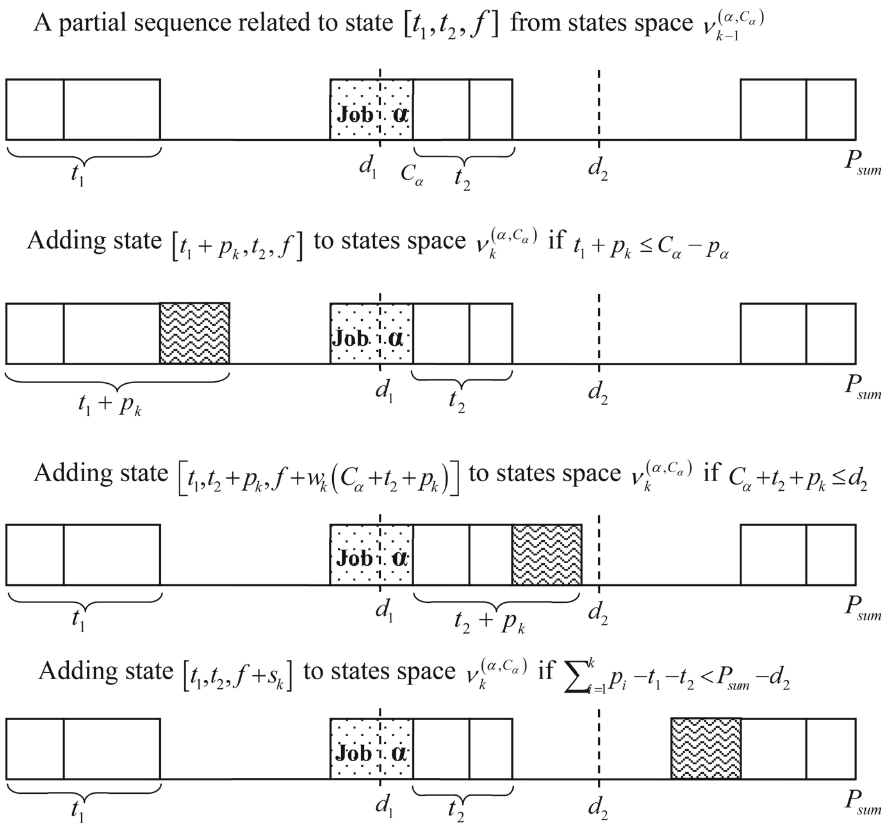
A partial sequence related to state $[t_1, t_2, f]$ from states space $\nu_{k-1}^{(\alpha,C_\alpha)}$
Job α
t_1
d_1
C_α
t_2
d_2
P_{sum}
Adding state $[t_1 + p_k, t_2, f]$ to states space $\nu_k^{(\alpha,C_\alpha)}$ if $t_1 + p_k \le C_\alpha - p_\alpha$
$t_1 + p_k$
Adding state $[t_1, t_2 + p_k, f + w_k(C_\alpha + t_2 + p_k)]$ to states space $\nu_k^{(\alpha,C_\alpha)}$ if $C_\alpha + t_2 + p_k \le d_2$
$t_2 + p_k$
Adding state $[t_1, t_2, f + s_k]$ to states space $\nu_k^{(\alpha,C_\alpha)}$ if $\sum_{i=1}^{k} p_i - t_1 - t_2 < P_{sum} - d_2$

Fig. 5 Details of creating new states in DP1

Now, we describe the approach of implementing lower bounds in DP1. Suppose job k is going to be included into group of early jobs from state $[t_1, t_2, f]$. If (15) holds, we can eliminate the new state. Values δ_T and δ_L denote remaining time intervals for scheduling tardy and lost jobs, respectively, and are calculated as (16). Value for Z_{heu} comes from algorithm MTLR.

$$f + \mathrm{LB}_T\,[\delta_T] + \mathrm{LB}_L\,[\delta_L] \geq Z_{heu} \tag{15}$$

$$\begin{aligned}
\delta'_L &= P_{\text{sum}} - \max\{C_\alpha, d_2\} - \sum_{i=1}^{k-1} p_i - t_1 - t_2 \\
\delta'_T &= d_2 - C_\alpha - t_2 + \max\left\{-\delta', 0\right\} \\
\delta_L &= \max\left\{\delta'_L, 0\right\} \\
\delta_T &= \max\left\{\delta'_T, 0.\right\}
\end{aligned} \tag{16}$$

To calculate the time complexity of DP1, consider the maximum number of states in each state space $\nu_k^{(\alpha,C_\alpha)}$ as $d_1.(d_2 - d_1)$. Step 1.2 iterates by a factor of total feasible states, $\sum_{k=1}^{n-1}\left|\nu_k^{(\alpha,C_\alpha)}\right|$, and is $O\left(nd_1\left(d_2 - d_1\right)\right)$. Similarly, we can show that complexity of step

1.3 is $O\left(d_1\left(d_2-d_1\right)\right)$. Step 1 iterates at most $P_{\text{sum}}=\sum_{i=1}^{n} p_i$ times by selecting different α and C_α values, and its complexity is $O\left(nP_{\text{sum}}d_1\left(d_2-d_1\right)\right)$. Finally, considering that step 2 is implemented in $O\left(P_{\text{sum}}\right)$, we conclude the overall time complexity of DP1 as $O\left(nP_{\text{sum}}d_1\left(d_2-d_1\right)\right)$.

5.2 Algorithm DP2

The basis of this algorithm is similar to DP1 and their only difference is in selecting straddling job. In contrast with the previous dynamic programming, we do not need to predefine a job as straddling at each iteration, and DP2 is able to distinguish suitable straddling job in each iteration of the algorithm. This algorithm considers the straddling job as a member of first group (group of early jobs) that makes minimum penalty if finishes at time C_α.

During an iteration, straddling job may change, and we can only determine the correct straddling job after scheduling all jobs. Algorithm DP2 keeps at most two states with equal t_1 and t_2 values in each iteration; the former, $[t_1, t_2, \alpha, f, f_{(\alpha)}]$, calculates minimum total penalty and the later, $[t_1, t_2, \alpha', f', f'_{(\alpha')}]$, gives minimum total penalty excluding straddling job. In this notation, f and $f_{(\alpha)}$ denote the total penalty and total penalty excluding straddling job, respectively.

In DP2, if a job is going to be scheduled in the group of early jobs and generates smaller penalty in comparison with the current straddling job, then the current straddling job will be considered as an early job and the new job takes the straddling place.

Let $[t_1, t_2, \alpha_{k-1}, f, f_{(\alpha_{k-1})}]$ be a state from the previous iteration, and DP2 is going to add job k to this state. Here, the following cases may arise, and we describe the new penalties f^{new} and $f^{new}_{(\alpha)}$ generated in each case.

1. Job k is early and the straddling job is not changed: $f^{\text{new}}=f$, $f^{\text{new}}_{(\alpha_k)}=f_{(\alpha_{k-1})}$.
2. Job k is early, the straddling job is changed from α_{k-1} to α_k and $C_\alpha \le d_2$: $f^{\text{new}}=f+\left(w_{\alpha_k}-w_{\alpha_{k-1}}\right)\left(C_\alpha-d_1\right)$, $f^{\text{new}}_{(\alpha_k)}=f_{(\alpha_{k-1})}$.
3. Job k is early, the straddling job is changed from α_{k-1} to α_k and $C_\alpha > d_2$: $f^{\text{new}}=f+s_{\alpha_k}-s_{\alpha_{k-1}}$, $f^{\text{new}}_{(\alpha_k)}=f_{(\alpha_{k-1})}$.
4. Job k is tardy: $f^{\text{new}}=f+w_k\left(C_\alpha+t_2+p_k-d_1\right)$, $f^{\text{new}}_{(\alpha_k)}=f_{(\alpha_{k-1})}+w_k\left(C_\alpha+t_2+p_k-d_1\right)$.
5. Job k is lost : $f^{\text{new}}=f+s_k$, $f^{\text{new}}_{(\alpha_k)}=f_{(\alpha_{k-1})}+s_k$.

Steps of this algorithm are as follows.

1. For each $C_\alpha \in [d_1+1, \min\left(d_1+p_{\max}, P_{\text{sum}}\right)]$.
   1.1. Set $\nu_0^{(C_\alpha)}=\{[0,0,0,0,0]\}$.
   1.2. For each $k \in \{1,2,\ldots,n\}$ $t_1 \in \{0,1,\ldots,C_\alpha\}$ and each $t_2 \in \left\{0,1,\ldots,\max\left(0, d_2-C_\alpha\right)\right\}$ consider all states $[t_1, t_2, \alpha_{k-1}, f, f_{(\alpha_{k-1})}]$ and $[t_1, t_2, \alpha'_{k-1}, f', f'_{(\alpha'_{k-1})}]$ in $\nu_{k-1}^{(C_\alpha)}$.

   1.2.1 (job k is early) If $t_1+p_k \le C_\alpha$ and DR1 and DR3 do not eliminate new state for job k,

   - If straddling job is not changed, add states $[t_1+p_k, t_2, \alpha_{k-1}, f, f_{(\alpha_{k-1})}]$ and $[t_1+p_k, t_2, \alpha'_{k-1}, f', f'_{(\alpha'_{k-1})}]$ to state space $\nu_k^{(C_\alpha)}$.
   - If $C_\alpha \le d_2$ and straddling job changes, add states $[t_1+p_k, t_2, \alpha_k, f+(w_{\alpha k}-w_{\alpha_{k-1}})(C_\alpha-d_1), f_{(\alpha_{k-1})}]$ and $[t_1+p_k, t_2, \alpha'_k, f+(w_{\alpha' k}-w_{\alpha'_{k-1}})\left(C_\alpha-d_1\right), f'_{(\alpha'_{k-1})}]$ to state space $\nu_k^{(C_\alpha)}$.

- If $C_\alpha > d_2$ and straddling job changes, add states $[t_1 + p_k, t_2, \alpha_k, f + s_{\alpha_k} - s_{\alpha_{k-1}}, f_{(\alpha_{k-1})}]$ and $[t_1 + p_k, t_2, \alpha'_k, f + s_{\alpha' k} - s_{\alpha'_{k-1}}, f'_{(\alpha'_{k-1})}]$ to state space $\nu_k^{(C_\alpha)}$.

1.2.2 (job k is tardy) If $C_\alpha + t_2 + p_k \leq d_2$ and $\sum_{i=1}^{k} p_i - t_1 \leq P_{\text{sum}} - C_\alpha$ and also DR1, DR2, LB1, and LB2 do not eliminate the new state for job k, then add states $[t_1, t_2 + p_k, \alpha_{k-1}, f + w_k (C_\alpha + t_2 + p_k - d_1), f_{(\alpha_{k-1})} + w_k (C_\alpha + t_2 + p_k - d_1)]$ and $[t_1, t_2 + p_k, \alpha'_{k-1}, f' + w_k (C_\alpha + t_2 + p_k - d_1), f'_{(\alpha'_{k-1})} + w_k (C_\alpha + t_2 + p_k - d_1)]$ to $\nu_k^{(C_\alpha)}$.

1.2.3 (job k is lost) If $\sum_{i=1}^{k} p_i - t_1 - t_2 < P_{\text{sum}} - d_2$ and $\sum_{i=1}^{k} p_i - t_1 \leq P_{\text{sum}} - C_\alpha$ and also DR2, DR3, LB1, and LB2 do not eliminate the new state for job k, then add states $[t_1, t_2, \alpha_{k-1}, f + s_k, f_{(\alpha_{k-1})} + s_k]$ and $[t_1, t_2, \alpha'_{k-1}, f' + s_k, f'_{(\alpha'_{k-1})} + s_k]$ to $\nu_k^{(C_\alpha)}$.

1.2.4 For all the states $[t_1, t_2, \alpha_k, f, f_{(\alpha_k)}] \in \upsilon_k^{(C_\alpha)}$ with equal values for t_1 and t_2, keep at most one state having the minimum value of f and for all the states $[t_1, t_2, \alpha'_k, f', f'_{(\alpha'_k)}] \in \upsilon_k^{(C_\alpha)}$ with equal values for t_1 and t_2, keep at most one state having the minimum value of $f'_{(\alpha'_k)}$.

1.2.5 Delete state space $\nu_{k-1}^{(C_\alpha)}$.

1.3. Set $Z^*_{C_\alpha} = \min\left(\min_{[t_1, t_2, \alpha_n, f, f_{(\alpha_n)}] \in \nu_n^{(C_\alpha)}} \{f\}, \min_{[t_1, t_2, \alpha'_n, f', f'_{(\alpha'_n)}] \in \nu_n^{(C_\alpha)}} \{f'\}\right)$.

2. Return $Z = \min_{C_\alpha} Z^*_{C_\alpha}$ as optimal solution.

While implementing the dominance rules in DP2, it must be noted that we cannot use them on straddling jobs, because these jobs may change during step 1.2.1. To examine lower bounds, suppose, job $k$is considered to be added into group of early jobs from a state $[t_1, t_2, \alpha_{k-1}, f, f_{(\alpha_{k-1})}]$. If (17) holds, we discard the new state. Here, $Z^{\min}_{C_\alpha}$ denotes minimum penalty caused by an unknown straddling job that completes at time C_α and is calculated by (18). From the fact that, in DP2, there is no need to prefix a job as straddling, its complexity is $O(n)$ times less than DP1, and, hence, is $O\left(P_{\text{sum}} d_1 (d_2 - d_1)\right)$.

$$f_{(\alpha_{k-1})} + Z^{\min}_{C_\alpha} + \text{LB}_T[\delta_T] + \text{LB}_L[\delta_L] \geq Z_{\text{heu}} \tag{17}$$

$$Z^{\min}_{C_\alpha} = \begin{cases} \min_i \{w_i\} . (C_\alpha - d_1) & \text{if } C_\alpha \leq d_2 \\ \min_i \{s_i\} & \text{if } C_\alpha > d_2. \end{cases} \tag{18}$$

6 Branch-and-bound algorithm

This section introduces a branch and bound algorithm for problem $1|d_i^1 = d_1, d_i^2 = d_2|\text{TL}$. This algorithm prefixes a job as straddling as well as its completion time, and, then, schedules other jobs into groups of early, tardy, and lost (groups 1–3, respectively) based on depth first search tree. Jobs are renumbered and selected for scheduling by WSPT order.

Figure 6 shows the depth first tree used for problem $1|d_i^1 = d_1, d_i^2 = d_2|\text{TL}$ with four jobs, where job 2 is straddling. Each level is related to one job, excluding the straddling job, and two numbers in each node show order of creating nodes and the group number, in which job is added, respectively. According to this figure, we first add job 1 to group 3, and then, jobs 3 and 4 are added to group 3, respectively.

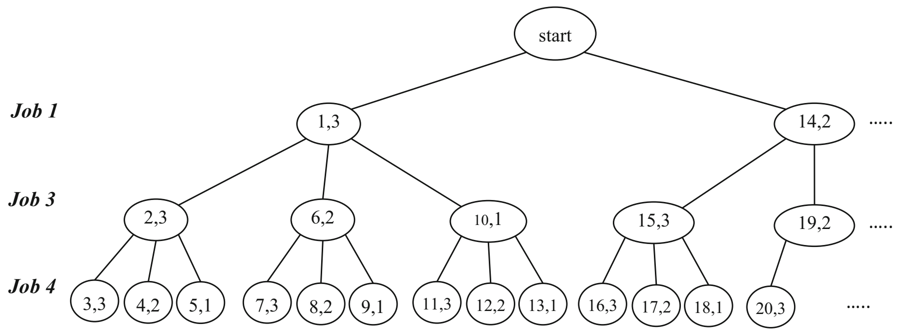

Fig. 6 Part of the search tree for algorithm BB1

Details of this branch-and-bound algorithm are described as follows. Variables t_1, t_2, and t_3 show total process time of jobs in groups 1–3. In this algorithm, we first assign jobs to group 3 and then add them to groups 2 and 1, because experiments show this type of assignment increases $\hat{Z}$ rapidly and improves the efficiency of lower bounds.

1. Calculate the upper bound *UB* from algorithm MTLR. For each $\alpha = 1, 2, \ldots, n$ and each $C_\alpha = [d_1 + 1, \min\{d_1 + p_\alpha, P_{\text{sum}}\}]$.
   1.1. Set $t_1 = t_2 = t_3 = 0$ and if $C_\alpha \leq d_2$, then $\hat{Z} = w_\alpha (C_\alpha - d_1)$ else $\hat{Z} = s_\alpha$.
   1.2. Assign jobs to a depth first tree of Fig. 6.

   1.2.1 If current job $\hat{i}$ is a candidate for group 1.

   - If $t_1 + p_{\hat{i}} > C_\alpha - p_\alpha$, then fathom current node and return to step 1.2.
   - Check DR1 and DR2 for job $\hat{i}$ and other tardy or lost jobs. If current node is fathomed, then return to step 1.2 else set $t_1 = t_1 + p_i$.

   1.2.2. If current job $\hat{i}$ is a candidate for group 2.

   - If $C_\alpha + t_2 + p_{\hat{i}} > d_2$ or $C_\alpha + t_2 + t_3 + p_i > P_{\text{sum}}$, then fathom current node and return to step 1.2.
   - Check LB1 and LB2 as well as DR1 and DR3 for job $\hat{i}$ and other early or lost jobs. If current node is fathomed, then return to step 1.2.
   - Set $t_2 = t_2 + p_{\hat{i}}$ and $\hat{Z} = \hat{Z} + w_{\hat{i}} (C_\alpha + t_2 - d_1)$.

   1.2.3 If current job $\hat{i}$ is a candidate for group 3.

   - If $t_3 \geq P_{\text{sum}} - d_2$, then fathom current node and return to step 1.2.
   - Check LB1 and LB2 as well as DR2 and DR3 for job $\hat{i}$ and other early or lost jobs. If current node is fathomed, then return to step 1.2.
   - Set $t_3 = t_3 + p_{\hat{i}}$ and $\hat{Z} = \hat{Z} + s_{\hat{i}}$.

   1.3 If a complete schedule with total penalty $\hat{Z}$ is achieved and $\hat{Z} < UB$, then set $UB = \hat{Z}$ and save this schedule as the best schedule found until now.

2. Return the final UB as optimal solution.

We can use the idea of letting the branch-and-bound algorithm to choose straddling job, but experiments indicate this idea increases run times in all instances, and hence, here, we discard this idea.

Table 3 Properties related to 32 groups of instances

Group	τ_1	τ_2	p	w	u	Group	τ_1	τ_2	p	w	u	Group	τ_1	τ_2	p	w	u
G1	L	L	L	L	L	G12	L	H	L	H	H	G23	H	L	H	H	L
G2	L	L	L	L	H	G13	L	H	H	L	L	G24	H	L	H	H	H
G3	L	L	L	H	L	G14	L	H	H	L	H	G25	H	H	L	L	L
G4	L	L	L	H	H	G15	L	H	H	H	L	G26	H	H	L	L	H
G5	L	L	H	L	L	G16	L	H	H	H	H	G27	H	H	L	H	L
G6	L	L	H	L	H	G17	H	L	L	L	L	G28	H	H	L	H	H
G7	L	L	H	H	L	G18	H	L	L	L	H	G29	H	H	H	L	L
G8	L	L	H	H	H	G19	H	L	L	H	L	G30	H	H	H	L	H
G9	L	H	L	L	L	G20	H	L	L	H	H	G31	H	H	H	H	L
G10	L	H	L	L	H	G21	H	L	H	L	L	G32	H	H	H	H	H
G11	L	H	L	H	L	G22	H	L	H	L	H						

7 Computational results

In this section, we examine the results of the mathematical model, heuristic algorithm, dynamic programming, and branch-and-bound algorithms on a number of randomly generated test problems. Computational experiments were performed on Intel Core$^{\text{TM}}$ i7-2600 CPU 3.4GHz with 4 GB RAM. On this system, CPLEX 12 was used as a mixed integer programming solver and the algorithms were coded in Visual studio C++ 2008.

We generate random instances for $n \in \{30, 50, 75, 100, 150, 200, 250\}$. Processing times, p_i, and tardiness weights, w_i, are drown from uniform distributions [1, 10] or [1, 100]. Using the method proposed in (Kethley and Alidaee 2002), due date d_1 is selected from uniform distribution between $P_{\text{sum}}(1-\tau_1-0.5R_1)$ and $P_{\text{sum}}(1-\tau_1+0.5R_1)$, and d_2 is from uniform distribution between $d_1+(P_{\text{sum}}-d_1)(1-\tau_2-0.5R_2)$ and $d_1+(P_{\text{sum}}-d_1)(1-\tau_2+0.5R_2)$, where $\tau_1 \in \{0.4, 0.8\}$, $\tau_2 \in \{0.2, 0.8\}$, $R_1 = 0.2$, and $R_2 = 0.4$. Lost penalty steps, u_i, are generated from one of uniform distributions $U[0, 0.5w_i(d_2-d_1)]$ or $U[w_i(d_2-d_1), 3w_i(d_2-d_1)]$.

According to the above parameters, 32 instance groups are created by combining parameters p_i, w_i, τ_1, τ_2, and u_i. For any given number of jobs, n, 20 random test instances are generated in each instance group. Table 3 shows the characteristics of these groups were L and H which show low and high levels for each parameter, respectively.

Table 11 (see Appendix) shows a summary of computational results obtained from running proposed methods on test instances in groups G1 to G16. The first two columns show group number and the number of jobs in each instance. The next two columns indicate the average and maximum gap between heuristic solutions and optimal ones. The results show that in more than 95 % of cases, average and maximum gaps are less than 2 and 4 %, respectively.

The next two columns show the average run times and number of instances solved by MIP model. The model is implemented by CPLEX 12 and is able to optimally solve instances up to 50 jobs in 1 h time limit.

Results of algorithm DP1 are provided in columns 7 to 14. In column 9, the percentage of fathomed states by lower bounds and dominance rules out of all created states are given. This ratio varies between 35 and 65 % in different instance groups and DP1 is able to solve instances up to 50 jobs. Columns 15 to 22 of Table 11 contain the results of running algorithm DP2. These results indicate that DP2 is able to optimally solve all instances up to 250 jobs

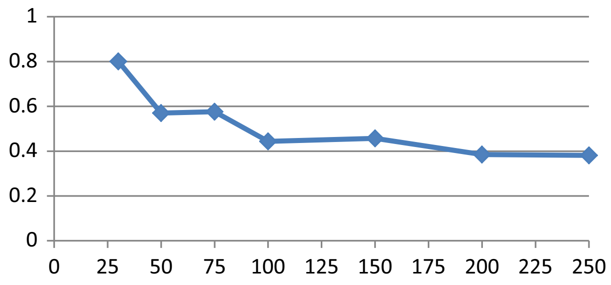

Fig. 7 Gap percentages for MTLR algorithm

Table 4 Problem parameters impact on MTLR error percentage

Value/variation	u	w	p	τ_2	τ_1
Low	0.74	0.64	0.49	0.49	0.69
High	0.46	0.56	0.71	0.71	0.51

with $p_i \in [1, 10]$ and instances up to 75 jobs with $p_i \in [1, 100]$ in reasonable times. The last columns in this table are about algorithm BB1 and show that this algorithm is able to solve instances up to 50 jobs with 55–65 % percent of fathomed nodes.

Figure 7 shows the average gap from optimal solution for algorithm MTLR in different instance sizes. This average error is less than 0.8 % and decreases by any increase in the size of instances. This shows good performance of MTLR in generating near optimal solutions for problem $1|d_i^1 = d_1, d_i^2 = d_2|\text{TL}$. Running times of this algorithm are less than 0.1 s in all the cases, and hence, are not reported.

In Table 4, we examine how changing the problem parameters influence optimality gap of algorithm MTLR. As it is seen, the algorithm performs better when the variation of processing times is low (i.e. $p_i \in [1, 10]$). By increasing the variation of w_i and u_i parameters, the average gap decreases, because of that the algorithm can categorize jobs into the groups E, T, and L more accurately under high variation of these two parameters.

Parameter τ_1 has no significant effect on the performance of heuristic algorithm, but when τ_2 increases, more number of jobs will be lost, and high penalties of the lost jobs cause an increase in optimality gap of algorithm MTLR.

Computational results show that all 30 job instances are solved by the mathematical model in average run time of 18.39 s. However, in the case of 50 job instances, the model is able to solve 298 instances out of 320 within 1 h time limit, and the average run time for solved instances is 1647.48 s.

Table 5 shows the impact of varying parameters on run times of mathematical model in 50 job instances. Variation of process times has a direct relation with run times, while increasing tardiness or lost penalties ($w_i's$ or $u_i's$) will decrease the model run times in average. Changing the parameter τ_1 has no significant effect on run times; however, results show a inverse relation between parameter τ_2 and average run times. That is because if τ_2 increases, the value of d_2 will decrease and less number of jobs can satisfy the fourth constraint of mathematical model, and hence, less number of jobs should be examined for laying in lost jobs' group.

Algorithm DP1 is able to solve instances up to 250 jobs with low variation of process times (i.e. $p_i \in [1, 10]$) and instances up to 50 jobs when process times are generated from $U[1, 100]$. Table 6 shows the number of solved instances out of 320, and Table 7 gives average of run times under low and high process time variation.

Figure 8 shows the percentage of fathomed nods by dominance rules and lower bounds in DP1. According to this figure, LB1 has the best performance in small-size instances; however,

Table 5 Problem parameters impact on CPLEX run times

Value/variation	u	w	p	τ_2	τ_1
low	1944.18	1739.57	1565.93	1924.44	1631.83
High	1350.77	1555.38	1729.02	1370.51	1663.12

Table 6 Number of solved instances out of 320 solved by DP1

	$n = 30$	$n = 50$	$n = 75$	$n = 100$	$n = 150$	$n = 200$	$n = 250$
Number of solved instances	320	313	160	160	160	157	92

Table 7 Problem parameters impact on DP1 run times

Process time variation	$n = 30$	$n = 50$	$n = 75$	$n = 100$	$n = 150$	$n = 200$	$n = 250$
Low	0.13	1.43	10.36	33.48	290.24	1105.53	2439.79
High	157.07	1447.97	–	–	–	–	–

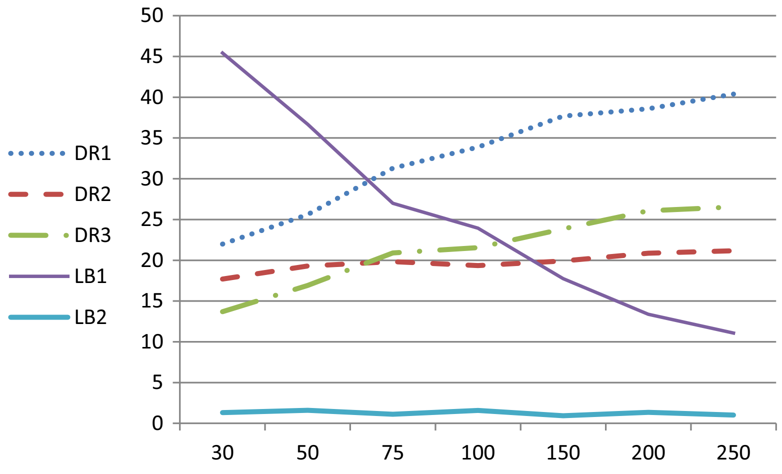

Fig. 8 Percentage of fathomed nodes in DP1

by increasing the size of instances, more number of nodes is fathomed by dominance rules in average. DR1 fathoms more nodes in comparison with other dominance rules, because it works on early and tardy groups and jobs in these two groups are scheduled before lost jobs.

Algorithm DP2 solves all instances up to 75 jobs. Based on Table 8, in groups with low variety of process times, DP2 is able to solve instances up to 250 jobs in less than 32 s. In case of high variety for process times, instances up to 75 jobs are solved in average 183 s, but larger instances are not solved due to system memory limitations.

Figure 9 shows the percentage of fathomed nodes by dominance rules and lower bounds in DP2. Here, LB2 has the best performance on instances with less than 100 jobs; however, by increasing the size of instances, more percentage of nodes is fathomed by dominance rules.

Table 8 Problem parameters impact on DP2 run times

Process time variation	$n = 30$	$n = 50$	$n = 75$	$n = 100$	$n = 150$	$n = 200$	$n = 250$
Low	0.02	0.07	0.29	0.84	4.96	12.67	31.19
High	11.09	60.63	182.89	–	–	–	–

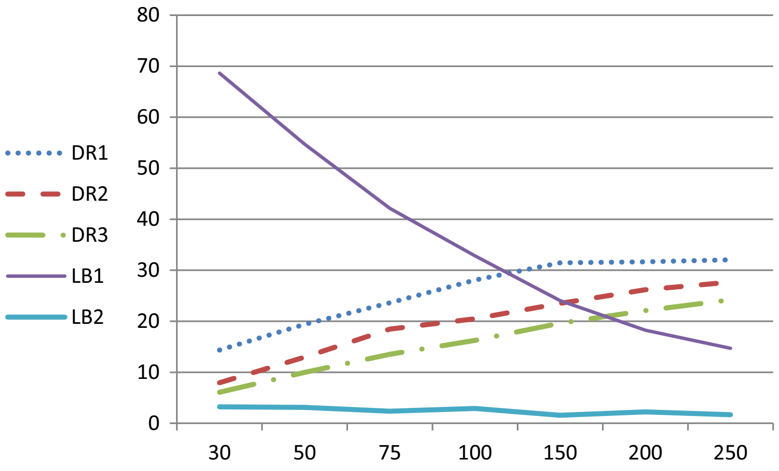

Fig. 9 Percentage of fathomed nodes in DP2

Table 9 Number of solved instances out of 320 solved by BB1

	$n = 30$	$n = 50$	$n = 75$	$n = 100$	$n = 150$	$n = 200$	$n = 250$
Number of solved instances	320	294	197	100	0	0	0

Table 10 Problem parameters impact on BB1 run times

Process time variation	$n = 30$	$n = 50$	$n = 75$	$n = 100$	$n = 150$	$n = 200$	$n = 250$
Low	0.28	49.76	1099.78	1941.2	–	–	–
High	9.93	1025.23	2754.15	2178.65	–	–	–

This shows the efficiency of LB1 in small- and medium-size instances and dominance rules in large-size instances.

Tables 9 and 10 give the number of solved instances by BB1 and average run times for instances with low and high variation of process times. For instances with $p_i \in [1, 10]$, algorithm BB1 is able to solve all instances up to 50 jobs in average run time 49.76 s. In case $p_i \in [1, 100]$, all instances up to 30 jobs are solved in average 9.39 s.

The percentage of fathomed nodes in BB1 is shown in Fig. 10, where LB1 has the best performance on fathoming nodes and such a way that it cuts more than 60 % of nodes in all problem sizes.

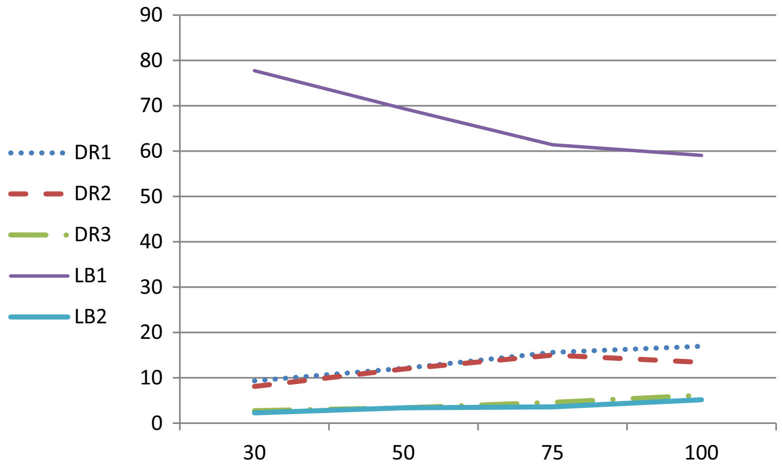

Fig. 10 Percentage of fathomed nodes in BB1

8 Conclusions

In this paper, we studied minimizing Tardy/Lost penalties on a single machine with common due dates. We examined time complexity of the problem and proposed a MIP model which classifies jobs into four groups of early, tardy, straddling, and lost jobs. Then, a heuristic algorithm was developed and, later, was used as upper bound in dynamic programming and branch-and-bound algorithms.

In Sect. 4, we introduced three dominance rules and two lower bounds, and in the next section, two dynamic programming algorithms were developed. These DPs are pseudo-polynomial with time complexities $O\left(nP_{\text{sum}}d_1\left(d_2-d_1\right)\right)$ and $O\left(P_{\text{sum}}d_1\left(d_2-d_1\right)\right)$. Then, we proposed a branch-and-bound algorithm based on depth first search tree.

To evaluate the proposed methods, we generated 32 groups of test instances with 30–250 jobs. Experiments indicated that increasing the variety of process times will make instances harder and cause algorithms need more time to solve them. For instances with 75 jobs or less, LB1 shows the best performance in cutting down search trees; but with increasing the size of instances, dominance rules show a better performance in comparison with LBs.

Average optimality gap of MTLR algorithm was less than 2 % for all instances, which proves the efficiency of this algorithm in finding near optimal solutions for problem $1|d_i^1=d_1, d_i^2=d_2|\text{TL}$. In overall, DP2 is the best algorithm for finding optimal solutions. This algorithm solves all instances with low variety of process times in less than 32 s and for high process time varieties, the algorithm solves all instances up to 75 jobs in less than 183 s.

The proposed Tardy/Lost penalty function can be considered as a general form for some well-known performance measures, such as late work, weighted tardiness, and tardiness with order acceptance assumption. Therefore, efficient solution approaches can also be applied in the case of these problems. As future researches, we suggest using meta-heuristics or constraint programming methods.

Appendix

See Table 11.

Table 11 Summary of running proposed methods on test instances

Group	n	MTLR		CPLEX		DP1							
		Error %		Ave. run time	No. optimal	Ave. run time	No. optimal	No. fathomed states	Percentage of fathomed states by				
		Ave.	Max.						DR1	DR2	DR3	LB1	LB2
G1	30	0.2	1.2	84.9	10	0.2	10	56.1	24.1	10.7	10.6	54	0.6
	50	0.8	2.6	1933.8	10	2.1	10	50.1	32.4	10.9	15.7	39.8	1.2
	75	0.8	1.9	0	0	20.6	10	43.8	37.3	12.8	17.4	31	1.4
	100	0.4	1	0	0	42.5	10	44.5	40.3	12	16.2	31.1	0.5
	150	0.7	1.2	0	0	486.9	10	39.7	46.8	11.2	19.4	22.1	0.5
	200	0.8	1.4	0	0	1726.3	10	38.5	50.5	10.8	20.5	17.3	0.9
	250	0.5	1.2	0	0	3486.9	1	38.1	52.9	12.4	21.7	12.9	0
G2	30	0.1	1	30.8	10	0.1	10	58.5	28.2	9.3	10.3	50.9	1.3
	50	0.3	2.5	2486.7	9	1.2	10	59.3	27.3	6.9	14.2	51.5	0.1
	75	0.3	1.4	0	0	9.5	10	52.8	35	11	15.2	38.1	0.8
	100	0.2	1	0	0	30	10	49.2	38.4	9.2	16.5	35.6	0.2
	150	0.5	1.6	0	0	285.6	10	45.5	45.1	8.6	17.2	28.6	0.4
	200	0.3	0.8	0	0	1227.8	10	42.4	51.8	7.8	16.6	23.8	0
	250	0.6	1.6	0	0	3369.8	5	40.9	55.4	8.9	16	19.2	0.5
G3	30	0.6	2.7	47.2	10	0.2	10	58	22.9	16.6	11.8	44.5	4.2
	50	0.5	1.6	1719.6	9	1.7	10	50.4	32.6	17.4	13	31.6	5.5
	75	0.4	0.9	0	0	10.3	10	49.9	33.5	16.6	13.3	32.5	4.1
	100	0.5	1.2	0	0	29.9	10	46.8	35.5	16.6	14.3	27.3	6.3
	150	0.7	1.8	0	0	373.1	10	44.4	41.8	19.6	18.6	16.8	3.2
	200	0.4	1.4	0	0	1406.2	10	42.2	47.7	18.2	18.5	11.3	4.3
	250	0.5	1	0	0	3485.9	4	41.5	49.1	18.6	19.3	9.6	3.4

Table 11 continued

Group	n	MTLR		CPLEX		DP1							
		Error %		Ave. run time	No. optimal	Ave. run time	No. optimal	No. fathomed states	Percentage of fathomed states by				
		Ave.	Max.						DR1	DR2	DR3	LB1	LB2
G4	30	0.5	4	11.3	10	0.1	10	60.1	24.9	9.6	8.2	56.1	1.1
	50	0	0.3	1247	10	1.1	10	57.9	30.2	12.9	12	41.4	3.4
	75	0.6	2.6	0	0	8.1	10	55.2	31.4	15.4	14.3	36.9	2
	100	0.2	0.9	0	0	22.1	10	53.2	35.8	14.7	13.3	32.7	3.5
	150	0.2	1.6	0	0	249.6	10	48.1	44.6	14.2	15.6	24.7	0.9
	200	0.1	0.4	0	0	1072.6	10	47.4	48.5	14.5	16.5	16	4.6
	250	0.1	0.4	0	0	2623.5	9	43.5	55	12.9	15.3	12.6	4.2
G5	30	1.3	3.5	74.3	10	287.5	10	59.5	25.8	11	11	50.6	1.6
	50	0.4	1.5	1646.5	10	2065.6	10	59.8	28.2	12.3	11.4	47.8	0.3
	75	0.4	1	0	0	–	–	–	–	–	–	–	–
G6	30	1.9	13.3	16.9	10	320.8	10	61.9	29.9	5.6	6.2	58	0.3
	50	0.2	0.8	2335.7	9	2133.4	10	62.1	32.2	8.8	8.6	49	1.4
	75	1.1	4	0	0	–	–	–	–	–	–	–	–
G7	30	0.9	3	68.3	10	341.7	10	60.1	21.9	13.7	7.8	51.9	4.7
	50	0.7	3.1	2142.2	9	2095.6	9	57.7	26.5	16.4	10.7	38.7	7.7
	75	0.7	2.3	0	0	–	–	–	–	–	–	–	–
G8	30	0	0.3	11.6	10	319.2	10	62.8	26.3	8.9	6.9	56.4	1.5
	50	0.3	1.6	1846.4	9	1631.1	10	61.2	24.4	8.6	6.2	54.6	6.2
	75	0.5	2	0	0	–	–	–	–	–	–	–	–
G9	30	0.4	3	23.2	10	0	10	54	15	12.2	16.4	56.4	0
	50	1.3	3.1	1631.7	9	0.3	10	50.4	13.7	13.7	20.8	51.6	0.3
	75	0.9	2	0	0	5.4	10	44.3	27	15.8	21.5	35.7	0
	100	1	2.3	0	0	22.2	10	43.4	23.3	15.2	23	38.5	0

Table 11 continued

Group	n	MTLR		CPLEX		DP1							
		Error %		Ave. run time	No. optimal	Ave. run time	No. optimal	No. fathomed states	Percentage of fathomed states by				
		Ave.	Max.						DR1	DR2	DR3	LB1	LB2
	150	0.8	1.7	0	0	132.3	10	40.4	29.6	14.5	25.8	30.1	0
	200	0.7	1.2	0	0	614.8	10	38.6	30.1	15.9	29.6	24.4	0
	250	0.5	1.2	0	0	1332	10	37.8	32.3	16.2	30.8	20.8	0
G10	30	1.7	8.6	5.6	10	0.1	10	53.2	16.4	6.8	20.7	56.1	0
	50	0.6	1.9	809.2	10	0.3	10	54.1	21.9	8.9	15.5	53.7	0
	75	0.2	0.7	0	0	3.2	10	47.9	23.7	8	21.7	46.5	0
	100	0.4	0.7	0	0	11.8	10	46.4	24.6	10.3	24	41.2	0
	150	0.2	0.6	0	0	74.1	10	42.8	31.7	9.2	23.4	35.6	0
	200	0.2	0.4	0	0	201.3	10	39	31.3	13.7	28.5	26.6	0
	250	0.1	0.4	0	0	743.3	10	36.8	39	10.1	26.4	24.4	0
G11	30	1.8	5.3	19.9	10	0	10	55.5	13.4	18	17.5	50.9	0.3
	50	1.1	3.4	2180.4	8	0.6	10	52.6	13.6	21.7	20.1	44.3	0.2
	75	1.1	2.1	0	0	4.3	10	47.7	18.3	24.3	24.2	33.2	0
	100	0.8	1.3	0	0	7.9	10	44.3	17.6	23	23.9	34.9	0.7
	150	0.8	2	0	0	70.2	10	40.7	23.7	22.7	24.7	27.8	1
	200	0.6	1.5	0	0	479.2	10	41.9	28.3	25.1	28.4	16.7	1.4
	250	0.5	1	0	0	541.6	10	41.1	24.8	25.3	29.8	19.8	0.3
G12	30	1	5.9	5.6	10	0	10	56.7	17.2	10.3	13.3	58.7	0.5
	50	0.4	1	323.5	10	0.4	10	50.5	17.9	10.4	15.4	56.2	0.1
	75	0.8	3.6	0	0	3.5	10	49.4	21.1	15.6	21.5	41.5	0.3
	100	0.4	1.5	0	0	11.2	10	45.9	27.9	15.9	19.8	35.2	1.2

Table 11 continued

Group	n	MTLR		CPLEX		DP1							
		Error %		Ave. run time	No.optimal	Ave. run time	No. optimal	No. fathomed states	Percentage of fathomed states by				
		Ave.	Max.						DR1	DR2	DR3	LB1	LB2
	150	0.2	0.6	0	0	83.5	10	43.1	35.2	15.6	20.6	27.6	0.9
	200	0.2	0.6	0	0	145.9	10	42.9	29.3	16	26.9	27.4	0.4
	250	0.2	0.8	0	0	927	10	40	41.1	17	22.7	18.7	0.4
G13	30	1.6	5.1	58.6	10	43.7	10	59	14.5	16.6	12.2	56.6	0
	50	1	1.9	2688.3	7	1100.4	10	58.2	18.1	17.9	19.9	44	0
	75	1.2	3.1	0	0	–	–	–	–	–	–	–	–
G14	30	2.1	7.1	10	10	58	10	54.5	20.2	6.3	16.8	55.7	1
	50	0.5	2.3	1078.2	9	506.5	10	56.4	20.8	11.1	14.9	53.2	0
	75	0.5	1.3	0	0	–	–	–	–	–	–	–	–
G15	30	1.4	5.1	39.2	10	86	10	59.8	14.1	23.7	13.5	48.4	0.3
	50	0.7	2.1	1600.3	8	592.2	10	55.8	14.9	24.2	18.9	41	1
	75	1	1.8	0	0	–	–	–	–	–	–	–	–
G16	30	0.6	2.2	4.6	10	42.1	10	55.3	17.2	6.8	8.6	67.4	0.1
	50	0.3	1.4	439.8	10	480.4	10	59.8	17.8	10.6	9.6	61.8	0.2
	75	0.6	1.6	0	0	–	–	–	–	–	–	–	–

Table 11 continued

Group	*n*	DP2								BB1							
		Ave. run time	No. optimal	No. fathomed states	Percentage of fathomed states by					Ave. run time	No. optimal	No. fathomed nodes	Percentage of fathomed nodes by				
					DR1	DR2	DR3	LB1	LB2				DR1	DR2	DR3	LB1	LB2
G1	30	0	10	53.7	15.8	3.7	4.9	74.4	1.2	0.1	10	61.6	12.1	3.8	3.2	80.3	0.6
	50	0.1	10	48.4	27.5	7	9.4	53.1	3	18	10	61.8	13.8	8.7	3.7	70.8	3.1
	75	0.5	10	44	33.4	10.8	14.5	38.7	2.6	1181.1	9	64.2	19.6	14.5	6.3	51.4	8.2
	100	1.1	10	44.2	36.3	10.9	13.9	38.1	0.7	2155.6	6	64.5	19	10.6	6.8	61.7	1.8
	150	7	10	42	42.5	12.1	20.2	24.5	0.8	0	0	–	–	–	–	–	–
	200	18.3	10	40.8	45.5	12.9	21.9	18.4	1.2	0	0	–	–	–	–	–	–
	250	48.7	10	41.9	43.2	18.3	24.9	13.6	0.1	0	0	–	–	–	–	–	–
G2	30	0	10	57	20.2	0.9	1.2	74.8	2.8	0.1	10	62.5	15	4.9	5.5	72.7	1.9
	50	0	10	57.5	23.3	1.1	1.8	73.5	0.3	1.2	10	64.7	16	3.8	5.3	74.8	0.1
	75	0.2	10	54.3	30.5	4.1	5.9	57.5	1.9	47.6	10	64.5	19.6	10.5	7	62.3	0.5
	100	0.7	10	53	33.5	4.4	7.9	53.7	0.5	553.3	10	64.9	22.7	9.9	10.8	56.5	0.1
	150	4	10	49.6	39.8	5.3	11.6	42.7	0.7	0	0	–	–	–	–	–	–
	200	11.4	10	48.5	44.4	6	15	34.6	0	0	0	–	–	–	–	–	–
	250	27.1	10	47.2	44.8	8.8	17.6	27.9	0.9	0	0	–	–	–	–	–	–
G3	30	0	10	53.1	12.8	7.3	4.2	66.8	8.9	0.2	10	61.6	12.7	6.2	2.8	71	7.3
	50	0.1	10	46.8	22.4	12.3	9.1	45.6	10.6	20.3	10	61.8	16.1	14.1	4.7	55	10.1
	75	0.2	10	48.7	25.8	12.6	8.9	45.2	7.5	685.9	9	64.2	18.3	12	3.6	62.3	3.6
	100	0.8	10	46.3	28.4	15.3	11.6	34.7	10	2002	6	65	18.7	13.4	4.9	48.5	14.6
	150	8.8	10	44.9	34.5	23.1	16.2	21.2	5	0	0	–	–	–	–	–	–
	200	15	10	44.4	38.6	24	17.4	14.2	5.9	0	0	–	–	–	–	–	–
	250	33.7	10	46	38.6	24.7	20	12.3	4.4	0	0	–	–	–	–	–	–

Table 11 continued

Group	*n*	DP2								BB1							
		Ave. run time	No. optimal	No. fathomed states	Percentage of fathomed states by					Ave. run time	No. optimal	No. fathomed nodes	Percentage of fathomed nodes by				
					DR1	DR2	DR3	LB1	LB2				DR1	DR2	DR3	LB1	LB2
G4	30	0.1	10	56.1	13.5	0.7	0.3	83.5	2.1	0.1	10	63.6	10.6	3.1	3.6	81.3	1.4
	50	0	10	55.5	20.8	3.2	2.1	67.5	6.3	6.3	10	63.5	16.6	7	4.6	66	5.8
	75	0.2	10	54.4	23.9	6.9	4.6	60.8	3.8	169	10	65.3	17.3	11.4	7.1	60.8	3.4
	100	0.6	10	53.5	28.2	7.2	5.4	52.6	6.6	1933.6	7	65.5	20.3	11.2	7.2	54.5	6.8
	150	5.9	10	51.1	35.6	9	9.5	44.4	1.5	0	0	–	–	–	–	–	–
	200	10.3	10	52.4	37.6	11.2	10.5	32.3	8.3	0	0	–	–	–	–	–	–
	250	21.1	10	51.2	40.5	11.9	15.8	24.4	7.4	0	0	–	–	–	–	–	–
G5	30	17.6	10	59.6	16.8	2.3	4.5	71.1	5.3	3.2	10	60.5	9.2	4	2.5	81.7	2.6
	50	82.2	10	57.3	21.9	6.3	7.4	63.9	0.5	384	10	63.8	12.2	5	2.2	80.2	0.3
	75	180.9	10	52.7	24.4	7.2	9.7	57.9	0.9	2799.3	3	63.8	17.4	8.1	2.5	70.8	1.2
G6	30	20	10	59.5	19.8	0.6	0.4	78.4	0.8	1.3	10	63	10.3	2.3	1.7	85.1	0.5
	50	78.2	10	59.5	23.6	1.1	1.4	68.4	5.5	56.3	10	64.2	14.5	6.2	4.2	72.7	2.4
	75	81.5	10	58.8	27	2.8	3.3	66	0.9	1507.4	10	63.5	16.8	5.9	3.4	71.3	2.6
G7	30	21.6	10	56.2	12	5	2.4	67.3	13.4	4.8	10	60.6	7.3	5	1	78.1	8.6
	50	71.8	10	56.8	14.7	9	5.6	61.8	8.9	561.2	9	61.2	13.1	8.2	2.3	58.3	18.1
	75	179.5	10	50.1	19.4	15.7	9.1	48.8	7	2795.3	4	62.5	19.1	8.8	2.3	64.4	5.3
G8	30	19.8	10	58.8	16.1	0.6	0.5	80.2	2.7	1.6	10	64.2	10.1	2.2	1.9	83.8	1.9
	50	66.1	10	55.7	16.3	1.6	0.8	71.7	9.6	112.6	10	64.9	11.4	3.7	2.3	75	7.6
	75	98.7	10	55.2	21.1	7.1	6.4	53.8	11.6	2182.7	5	64.2	15.7	6.4	5.2	67.1	5.6
G9	30	0	10	50.3	7.6	2	8.9	81.4	0	0.1	10	58.2	5.7	4.5	5.2	84.6	0
	50	0	10	45.8	9.1	5.7	12.9	71.7	0.6	0.8	10	58.7	6	7	6.4	79.8	0.7
	75	0.1	10	43.5	25.2	11.4	19	44.4	0	714.4	9	63.6	9.5	13.4	6.6	70.5	0

Table 11 continued

Group	n	DP2								BB1							
		Ave. run time	No. optimal	No. fathomed states	Percentage of fathomed states by					Ave. run time	No. optimal	No. fathomed nodes	Percentage of fathomed nodes by				
					DR1	DR2	DR3	LB1	LB2				DR1	DR2	DR3	LB1	LB2
	100	0.3	10	43.9	22.6	10.8	21.4	45.2	0	1176.2	9	61.4	9	11.9	7.5	71.5	0
	150	1.8	10	42.2	29.6	11.5	25.1	33.9	0	0	0	–	–	–	–	–	–
	200	6.3	10	41.7	30.3	13.7	30.3	25.8	0	0	0	–	–	–	–	–	–
	250	10.5	10	42	32.8	13.6	32.6	21.1	0	0	0	–	–	–	–	–	–
G10	30	0	10	50.9	10.8	1.4	7.8	80	0	0	10	57.4	6.1	1.5	6.7	85.7	0
	50	0.1	10	48.7	16.8	1.5	6.8	74.9	0	0.5	10	58.2	9.6	3.1	2.9	84.4	0
	75	0.1	10	49.7	18.4	2.6	8.1	70.8	0	14.5	10	61.7	13.5	8.7	18	59.7	0
	100	0.2	10	48.3	21.9	5.7	13.1	59.3	0	115.9	10	57.8	10.5	7.2	14.9	67.4	0
	150	1	10	46.7	31.3	5.8	16.9	46.1	0	0	0	–	–	–	–	–	–
	200	2	10	45.6	29.6	10.3	26.6	33.5	0	0	0	–	–	–	–	–	–
	250	6.2	10	45.4	35.5	7.3	27.4	29.8	0	0	0	–	–	–	–	–	–
G11	30	0	10	49.1	6.8	6.1	10.4	75.8	0.8	0	10	60.2	4.4	7.4	5.7	82.1	0.4
	50	0	10	48.6	8.8	11	12.8	67	0.5	1.8	10	60.5	6.5	10.9	6	76.5	0.2
	75	0.1	10	45.1	14.4	20.7	18.7	46.2	0.1	174.1	10	62.4	10.4	16.1	7.5	66	0
	100	0.2	10	42.6	14.1	18.3	21.9	44.4	1.5	1082.2	8	61	7.8	15.3	7.3	68.3	1.2
	150	1	10	43	19.8	19.2	26	33.1	2	0	0	–	–	–	–	–	–
	200	5.4	10	45.4	24.1	25.7	27.5	20.1	2.5	0	0	–	–	–	–	–	–
	250	4.4	10	45.6	21	23.9	30.1	24.4	0.6	0	0	–	–	–	–	–	–
G12	30	0	10	50.7	8.6	1.7	2.2	86.3	1.2	0	10	58.6	6.1	3.8	5.2	84.4	0.5
	50	0	10	50.6	11.3	2.3	5.6	80.6	0.3	2.1	10	62.2	7.3	5.7	8	78.9	0
	75	0.1	10	47.6	15.1	8.9	12.9	62.4	0.7	84.6	10	60.1	9.8	9.6	10.2	70.3	0.2

Table 11 continued

Group	*n*	DP2								BB1							
		Ave. run time	No. optimal	No. fathomed states	Percentage of fathomed states by					Ave. run time	No. optimal	No. fathomed nodes	Percentage of fathomed nodes by				
					DR1	DR2	DR3	LB1	LB2				DR1	DR2	DR3	LB1	LB2
	100	0.2	10	45.4	22	10.9	15.3	49	2.8	1006.5	9	62	12.2	12.3	13.5	60	2
	150	1.3	10	45.9	28.1	13	17	40.2	1.8	0	0	–	–	–	–	–	–
	200	1.5	10	46.5	22.8	12.5	19.7	44.1	0.9	0	0	–	–	–	–	–	–
	250	8.5	10	47.7	31.9	14.9	24.7	27.7	0.7	0	0	–	–	–	–	–	–
G13	30	2.7	10	55.3	7.7	2.3	4.6	85.4	0	0.8	10	60.2	3.6	4.5	1.3	90.5	0
	50	37.9	10	54.7	14	7.1	13.4	65.4	0.1	342.5	10	63.6	6.7	7.7	4.3	81.3	0
	75	142.6	10	45.9	19.5	12.6	17.6	50.1	0.2	2190.7	5	61.8	6.7	10	4.5	78.9	0
G14	30	3.4	10	51.1	13.9	1	3.1	79	3.1	0.3	10	55.6	5.2	1.3	1.8	90.3	1.3
	50	19.7	10	53.6	14.4	2	4.8	78.8	0	26.6	10	58.7	6.6	4.7	4.3	84.4	0
	75	24.7	10	54.5	19.6	2.2	5.4	72.7	0	622.1	9	59.7	7.6	5.4	8.4	78.6	0
G15	30	5	10	56.6	8.5	5.8	7.4	77.4	0.8	1.8	10	60.7	5.4	9.3	2.7	82.3	0.3
	50	22.5	10	51.1	9.7	12.4	12.9	62.7	2.3	358	10	62.4	5.7	9.6	4.1	79.1	1.4
	75	55.6	10	51.3	10.4	11.9	11	63.5	3.2	1276.9	8	63.2	7.5	9.1	3.5	77.8	2.1
G16	30	2.4	10	48.9	9.4	1	2.5	86.9	0.2	0.2	10	55.8	4.4	1.5	3.3	90.7	0.1
	50	18.3	10	56	11.2	1.9	1.6	84.9	0.4	12.7	10	61.5	5.9	2.9	3.1	88	0.2
	75	59	10	48.6	17.2	7.4	10.2	63.2	2	896.9	9	61.5	7.2	7.8	6.3	77.1	1.6

References

Baptiste P, Pape C (2005) Scheduling a single machine to minimize a regular objective function under setup constraints. Discrete Optim 2(1):83–99

Baptiste P, Sadykov R (2009) On scheduling a single machine to minimize a piecewise linear objective function: A compact MIP formulation. Naval Res Logist 56(6):487–502

Carrasco RA, Iyengar G, Stein C (2013) Single machine scheduling with job-dependent convex cost and arbitrary precedence constraints. Oper Res Lett 41(5):436–441

Chandra C, Liu Z, He J, Ruohonen T (2014) A binary branch and bound algorithm to minimize maximum scheduling cost. Omega 42(1):9–15

Chen B, Potts CN, Woeginger GJ (1998) A review of machine scheduling. In: Du DZ, Pardalos PM (eds) Handbook of combinatorial optimization. Kluwer Academic Publishers, Boston, pp 21–169

Cheng TCE, Ng CT, Yuan JJ, Liu ZH (2005) Single machine scheduling to minimize total weighted tardiness. Eur J Oper Res 165:423–443

Colin EC, Quinino RC (2005) An algorithm for insertion of idle time in the single-machine scheduling problem with convex cost functions. Comput Oper Res 32(9):2285–2296

Detienne B, Dauzère-Pérès S, Yugma C (2012) An exact approach for scheduling jobs with regular step cost functions on a single machine. Comput Oper Res 39(5):1033–1043

Engels DW, Karger DR, Kolliopoulos SG, Sengupta S, Uma RN, Wein J (2003) Techniques for scheduling with rejection. J Algorithms 49(1):175–191

Fathi Y, Nuttle HWL (1990) Heuristics for the Common due date weighted tardiness problem lIE. Transactions 22(3):215–225

Federgruen A, Mosheiov G (1994) Greedy heuristics for single-machine scheduling problems with general earliness and tardiness costs. Oper Res Lett 16:199–208

Kahlbacher HG (1993) Scheduling with monotonous earliness and tardiness penalties. Eur J Oper Res 64:258–277

Karakostas G, Kolliopoulos SG, Wang J (2009) An FPTAS for the minimum total weighted tardiness problem with a fixed number of distinct due dates. In: Ngo HQ (ed) Computing and combinatorics. Lecture Notes in Computer Science, vol 5609. Springer, Berlin Heidelberg, pp 238–248

Kellerer H, Strusevich VA (2006) A fully polynomial approximation scheme for the single machine weighted total tardiness problem with a common due date. Theor Comput Sci 369:230–238

Kethley RB, Alidaee B (2002) Single machine scheduling to minimize total weighted late work: a comparison of scheduling rules and search algorithms. Comput Ind Eng 43:509–528

Kianfar K, Moslehi G (2013) A note on "Fully polynomial time approximation scheme for the total weighted tardiness minimization with a common due date". Discrete Appl Math 161(13–14):2205–2206

Kolliopoulos SG, Steiner G (2006) Approximation algorithms for minimizing the total weighted tardiness on a single machine. Theor Comput Sci 355(3):261–273

Koulamas C (2010) The single-machine total tardiness scheduling problem: Review and extensions. Eur J Oper Res 202(1):1–7

Lawler EL (1964) On scheduling problems with deferral costs. Manag Sci 11(2):280–288

Lawler EL, Moore JM (1969) A functional equation and its application to resource allocation and sequencing problems. Manag Sci 16:77–84

Lenstra JK, Rinnoy-Kan AHG, Brucker P (1977) Complexity of machine scheduling problems. Ann Discrete Math 1:343–362

Leung JYT (2004) Minimizing total weighted error for imprecise computation tasks and related problems. In: Leung JYT (ed) Handbook of scheduling: algorithms models and performance analysis. CRC Press, Boca Raton, pp 3416–3431

Pinedo M (1995) Scheduling: theory algorithms and systems. Prentice Hall, New Jersey

Potts CN, Van Wassenhove LN (1992a) Approximation algorithms for schrduling a single machine to minimize total late work. Oper Res Lett 11(5):261–266

Potts CN, Van Wassenhove LN (1992b) Single machine scheduling to minimize total late work. Oper Res 40(3):586–595

Shabtay D (2008) Due date assignment and scheduling a single machine with a general earliness/tardiness cost function. Comput Oper Res 35:1539–1545

Shabtay D, Gaspar N, Kaspi M (2013) A survey on offline scheduling with rejection. J Sched 16(1):3–28

Slotnick SA (2011) Order acceptance and scheduling: A taxonomy and review. Eur J Oper Res 212(1):1–11

Sterna M (2011) A survey of scheduling problems with late work criteria. Omega 39:120–129

Yuan J (1992) The NP-hardness of the single machine common due date weighted tardiness problem. Syst Sci Math Sci 5:328–333

Zhou X, Cai X (1997) General stochastic single-machine scheduling with regular cost functions. Math Comput Model 26(3):95–108