Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 35 (2008) 827–844
computers & operations research
www.elsevier.com/locate/cor

Single-machine scheduling with an availability constraint to minimize the weighted sum of the completion times

Imed Kacem*, Chengbin Chu, Ahmed Souissi

Université de Technologie de Troyes, ICD, LOSI FRE CNRS 2848, 12 rue Marie Curie, BP 2060, Troyes, France

Available online 16 June 2006

Abstract

In this article, we consider a single-machine scheduling problem with one unavailability period, with the aim of minimizing the weighted sum of the completion times. We propose three exact methods for solving such a problem: a branch-and-bound method based on new properties and lower bounds, a mixed integer programming model, and a dynamic programming method. These methods were coded and tested on randomly generated instances, and their performances were analyzed. Our numerical experiments show that the branch-and-bound method and the dynamic programming method are complementary. Using these approaches, we are able to solve problems with up to 3000 jobs within a reasonable computation time.

Keywords: Scheduling; Availability constraint; Branch and bound; Integer programming; Dynamic programming

1. Introduction

In the production context, managing production activities efficiently by taking maintenance requirements into account is becoming more and more important, which explains why many researchers have become interested in the subject, particularly the problem of scheduling [24]. Grounded in real industrial problems, this paper focuses on scheduling a set of jobs on a single machine which must undergo maintenance. The aim is to minimize the sum of the weighted completion times, given that the machine is unavailable during a certain period. This is a valid objective because the weights can quantify the stocking cost per a unit of time of the products to transform. Thus, the weighted sum of the completion times can represent the global stocking cost.

This type of problem has been studied in the literature for different objective functions. Given the aim of our study, we provide a brief overview of previous works related to the minimization of the total (weighted) completion time.

The simplest model—minimizing total completion time with a single period of unavailability (denoted $1, h_1 \| \sum C_i$)—was proved to be NP-Hard by Adiri et al. [1] and Lee and Liman [2]. These authors also studied

* Corresponding author. Université de Technologie de Troyes, France. Tel.: +33 3 25 71 58 54; fax: +33 3 25 71 56 49.
E-mail addresses: kacem@utt.fr (I. Kacem), chu@utt.fr (C. Chu), souissi@utt.fr (A. Souissi).

0305-0548/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2006.04.010

the performance of the SPT (shortest processing time) rule which consists in scheduling the jobs in non-decreasing order according to their processing times. Lee and Liman [2] showed that the SPT rule has a tight worst-case error bound of $\frac{2}{7}$. Sadfi et al. [3] studied the same problem and proposed an improved version of the SPT rule (called MSPT); they proved that the worst-case performance ratio is $\frac{20}{17}$. Sadfi et al. [4] have also proposed an efficient dynamic programming algorithm for solving the same problem in $O(nR)$ time, where R is the start time of the unavailability period.

Qi et al. [5] studied the problem of simultaneously scheduling jobs and maintenance tasks on a single machine. They considered the minimization of the sum of completion times, proving that the problem is NP-hard in the strong sense. They proposed three heuristic algorithms and a branch-and-bound (BAB) algorithm to solve the problem. Recently, Chen [6] proposed a BAB method for solving a similar problem.

Lee and Liman [7] studied a two parallel machine scheduling problem with the aim of minimizing the total completion time when one machine is continuously available and the other machine is available only up to a given time. They proved that the problem is NP-hard and proposed a dynamic programming solution method. They also proposed an SPT-based heuristic and showed that the worst-case error bound is $\frac{1}{2}$.

Mosheiov [8] studied the scheduling problem on m identical parallel machines to minimize the total completion time for cases in which each machine M_j is available in interval $[x_j, y_j]$ with $0 \leqslant x_j < y_j$, and showed that the SPT rule is asymptotically optimal as the number of jobs moves towards infinity.

Lee [9] studied the problem on two parallel machines in an attempt to minimize the total weighted completion time with one period of unavailability on one of the machines. They solved both resumable $(P_2, h_{21}|r - a| \sum w_i C_i)$ and non-resumable $(P_2, h_{21} \| \sum w_i C_i)$ versions using a dynamic programming approach. Sadfi and Ouarda [10] studied the two-machine total completion time scheduling problem with one period of maintenance on each machine. They also solved the problem using a dynamic programming approach.

Wang et al. [11] studied the problem of minimizing the weighted sum of completion times, given an arbitrary number of unavailability periods and an assumption of preventive jobs, as well as the special case with only one period of unavailability. They showed that the problem is NP-hard and proposed two heuristic solution methods, which were evaluated for the worst case. Recently, Kacem and Chu [12] studied the problem $1, h_1 \| \sum w_i \cdot C_i$ and showed that both WSPT[1] and MWSPT[2] rules have a tight worst-case performance ratio of 3 under some conditions. Kacem et al. have also proposed a BAB method and a mixed-integer linear programming method for solving the same problem [13].

A few papers have considered flow-shop and job-shop environments. Aggoune has proposed one noteworthy approach [14] and has worked with Portmann to develop a heuristic method for solving the flow-shop problem [15]. For more details and more references concerning these specific environments, please refer to the recent state-of-the-art papers by Sanlaville [16] and Lee [17]. Additional information can be found in a paper by Türkcan (Ref. [18]).

In this article, we focus on the scheduling problem on a single machine with one unavailability period, with the aim of minimizing the weighted sum of the completion times. In our study, the unavailability period, or maintenance period, is known in advance. In the next section, we propose several mathematical properties and new lower bounds, which are analytically compared. Our results show that one of the lower bounds is always at least as tight as the other lower bounds. Despite what may appear to be a negative result, we felt it was very important to report all the lower bounds in order to show that some ideas are less productive than others. In Section 3, we propose three exact methods based on the results of our analysis. These methods were tested intensively on a large data set and the results were compared; these test results are reported in Section 4. Section 5 presents our conclusions and suggests ideas for future research.

2. Lower bounds and properties

In this section, we define several terms and formulate the problem to be studied. We also propose some mathematical properties and present several lower bounds. These bounds will be analytically compared, and the results of the analysis will be used to construct exact methods presented in Section 3.

P is used to denote the following scheduling problem: the scheduling of N jobs on a single machine by minimizing $F = \sum_i w_i \cdot C_i$, where each job i has a processing time p_i and a weight w_i. C_i denotes the completion time for job i.

[1] WSPT: Weighted shortest processing time.

[2] MWSPT: Modified weighted shortest processing time.

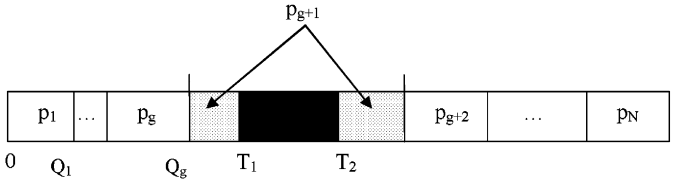


Fig. 1. Illustration of the WSRPT heuristic.

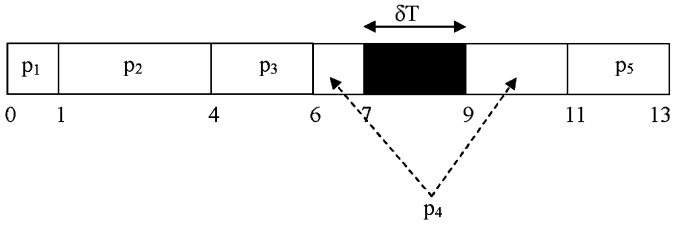


Fig. 2. Illustration of WSRPT rule for Example 1.

The machine is not available between T_1 and T_2. It is assumed that data are integers. Preemption is not allowed, unless the opposite is indicated. The preemptive version of the problem is denoted P_{pre}.

Definition 1. WSPT is the heuristic that schedules jobs in non-decreasing order according to (p_i/w_i).

If all the jobs can be inserted before T_1, the solution obtained using the WSPT rule is obviously optimal for problem P. For this reason, we consider only the problems in which all the jobs cannot be scheduled before T_1.

In the remainder of the paper, we assume, without loss of generality, that the jobs are renumbered in the order defined for WSPT (i.e., $p_1/w_1 \leqslant p_2/w_2 \leqslant \cdots \leqslant p_N/w_N$). $F^*(\pi)$ denotes the minimal weighted sum of the completion times for the problem π and $F_\sigma(\pi)$ denotes the weighted sum of the completion times of schedule σ for problem π. For the sake of simplicity, we define the following quantity, which is the sum of the processing times of all jobs through job k:

$$Q_k = \sum_{i=1}^{k} p_i. \tag{1}$$

Let job g be such that $Q_g \leqslant T_1$ and $Q_{g+1} > T_1$. Furthermore, let δ be the time between the completion of job g and the beginning of the period of unavailability:

$$\delta = T_1 - Q_g. \tag{2}$$

Definition 2. WSRPT is the heuristic that schedules jobs according to WSPT, except that job $g+1$, which cannot be completed before T_1, is interrupted and resumes after T_2 (see Fig. 1).

Example 1. This example is used to illustrate the WSPT and the WSRPT rules. Five jobs must be scheduled, such that $p_1 = 1$, $w_1 = 3$, $p_2 = 3$, $w_2 = 6$, $p_3 = 2$, $w_3 = 2$, $p_4 = 3$, $w_4 = 2$, $p_5 = 2$ and $w_5 = 1$; $T_1 = 7$ and $T_2 = 9$; and $p_1/w_1 \leqslant p_2/w_2 \leqslant p_3/w_3 \leqslant p_4/w_4 \leqslant p_5/w_5$.

Figs. 2 and 3 depict the schedules obtained by applying WSRPT and WSPT rules. In this example, we have $g+1=4$ and thus, the fourth job in the WSRPT schedule is interrupted.

2.1. Lower bound based on the WSRPT rule

Proposition 1 (*Wang [11], Lee [19]*). $F_{\text{WSRPT}}(P_{\text{pre}}) - F^*(P_{\text{pre}}) \leqslant w_{g+1} \cdot (T_2 - T_1)$.

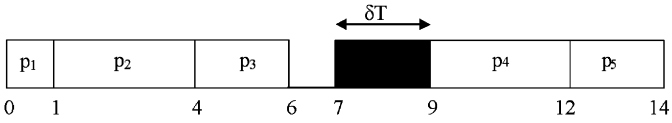


Fig. 3. Illustration of WSPT rule for Example 1.

Proposition 2. *Let* $LB_1 = F_{\mathrm{WSRPT}}(P_{\mathrm{pre}}) - w_{g+1} \cdot (T_2 - T_1)$. *Thus,* $LB_1 \leqslant F^*(P)$.

More specifically, LB_1 can be expressed as follows:

$$LB_1 = \sum_{i=1}^{g} w_i Q_i + \sum_{i=g+1}^{N} w_i (Q_i + \delta T) - w_{g+1} \cdot \delta T, \tag{3}$$

where $\delta T = T_2 - T_1$.

2.2. Lower bound based on a flexible scheduling of maintenance period

Definition 3. Consider a $w_t \geqslant 0$. Let P'_{w_t} denote a problem identical to problem P with the addition of a dummy job t (representing the unavailability period). This dummy job t has a processing time $p_t = \delta T$ and a weight w_t, and must start exactly at time T_1. This yields the following relation:

$$F^*(P'_{w_t}) = F^*(P) + w_t \cdot T_2. \tag{4}$$

Definition 4. RP'_{w_t} is the relaxed problem of P'_{w_t}, with an assumption that the dummy job t can start at any time. In practice, such a relaxation might mean that the period of preventive maintenance may be scheduled in a flexible rather than fixed manner. Obviously, we have

$$F^*(RP'_{w_t}) \leqslant F^*(P'_{w_t}). \tag{5}$$

Proposition 3. *Let* $LB_2(w_t) = F_{\mathrm{WSPT}}(RP'_{w_t}) - w_t \cdot T_2$ *for a given positive* w_t. *Thus,* $\forall w_t \geqslant 0$, $F^*(P) \geqslant LB_2(w_t)$.

Proof. According to Eq. (4), $F^*(P) = F^*(P'_{w_t}) - w_t \cdot T_2$. Given Eq. (5), it can be deduced that $F^*(P) = F^*(P'_{w_t}) - w_t \cdot T_2 \geqslant F^*(RP'_{w_t}) - w_t \cdot T_2$. In addition, since RP'_{w_t} can be solved optimally using WSPT, then this proposition is justified. □

Lemma 1. *WSPT is optimal for problem P if* $\delta = 0$, *i.e., if there is no gap between finishing a job and beginning maintenance* (*see Fig.* 4).

Proof. Consider the case in which $w_t = (\delta T / p_{g+1}) \cdot w_{g+1}$. Given this case, the optimal solution for RP'_{w_t} is not only feasible for the problem P'_{w_t}; it is also optimal for P'_{w_t}. Note that Lee proposed a similar result for the preemptive case [19]. □

2.2.1. A study of the function LB_2 according to w_t

In order to study the function LB_2 according to the variation of w_t, let us consider the case in which w_t belongs to the interval: $[(\delta T / p_{k+1}) \cdot w_{k+1}, (\delta T / p_k) \cdot w_k[$, where $k \in \{1, 2, \ldots, N-1\}$.

If $w_t \in]w_{k+1} \cdot \delta T / p_{k+1}, w_k \cdot \delta T / p_k[$, an optimal solution of RP'_{w_t} would be to schedule the jobs $1, 2, \ldots, N$ in the order of their index, inserting job t between jobs k and $k+1$ (see Fig. 5). The value of LB_2 would then be given by

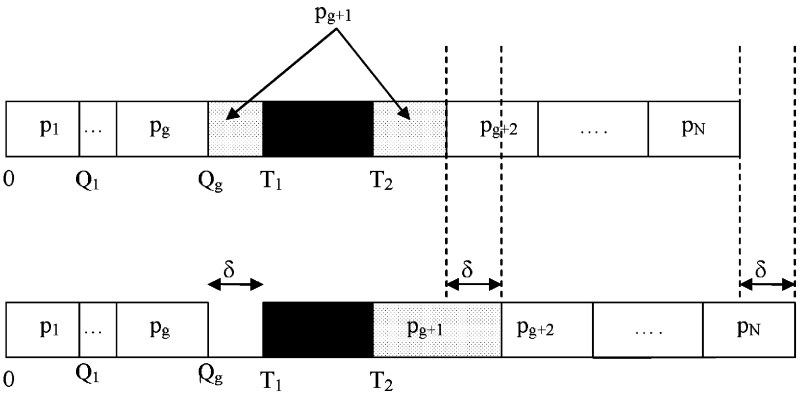


Fig. 4. Illustration of the WSPT and WSRPT rules.

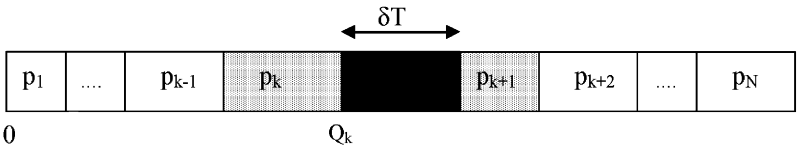


Fig. 5. Illustration of the LB_2 calculation.

the following equation:

$$\begin{aligned} LB_2(w_t) &= \sum_{i=1}^{k} w_i \cdot Q_i + \sum_{i=k+1}^{N} w_i \cdot (Q_i + \delta T) + w_t \cdot (Q_k + \delta T) - w_t \cdot T_2 \\ &= \sum_{i=1}^{k} w_i \cdot Q_i + \sum_{i=k+1}^{N} w_i \cdot (Q_i + \delta T) + w_t \cdot (Q_k - T_1). \end{aligned} \tag{6}$$

Therefore, it can be concluded that

$$\frac{\mathrm{d}LB_2}{\mathrm{d}w_t} = Q_k - T_1 \begin{cases} > 0 & \text{if } k \geqslant g+1, \\ \leqslant 0 & \text{if } k < g+1. \end{cases} \tag{7}$$

The function LB_2 is linear in the intervals $](\delta T/p_{k+1}) \cdot w_{k+1}, (\delta T/p_k) \cdot w_k[$. It increases with w_t if $k \geqslant g+1$ and decreases with w_t if $k < g+1$.

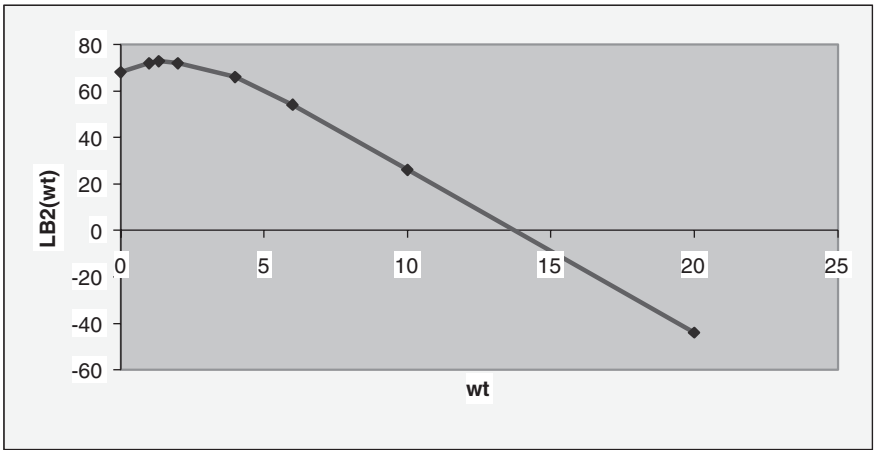
Consequently, LB_2 is a piecewise linear function that increases when $w_t \leqslant (\delta T/p_{g+1}) \cdot w_{g+1}$ and decreases when $w_t > (\delta T/p_{g+1}) \cdot w_{g+1}$.

It can therefore be deduced that the maximum for the function LB_2 is obtained when $w_t = (\delta T/p_{g+1}) \cdot w_{g+1}$.

From this result, the following lemma can be established.

Lemma 2. *The maximum for the function LB_2 is obtained when $w_t = (\delta T/p_{g+1}) \cdot w_{g+1}$ according to the following equation*:

$$\begin{aligned} LB_2^* &= \max_{w_t \geqslant 0} (LB_2(w_t)) = LB_2\left(\frac{\delta T}{p_{g+1}} \cdot w_{g+1}\right) \\ &= \sum_{i=1}^{g+1} w_i \cdot Q_i + \sum_{i=g+2}^{N} w_i \cdot (Q_i + \delta T) + \frac{\delta T}{p_{g+1}} w_{g+1}(Q_{g+1} - T_1). \end{aligned} \tag{8}$$



wt	LB2(wt)
0	68
1	72
1,333	72,666
2	72
4	66
6	54
10	26
20	-44

Fig. 6. Curve and values table for LB_2.

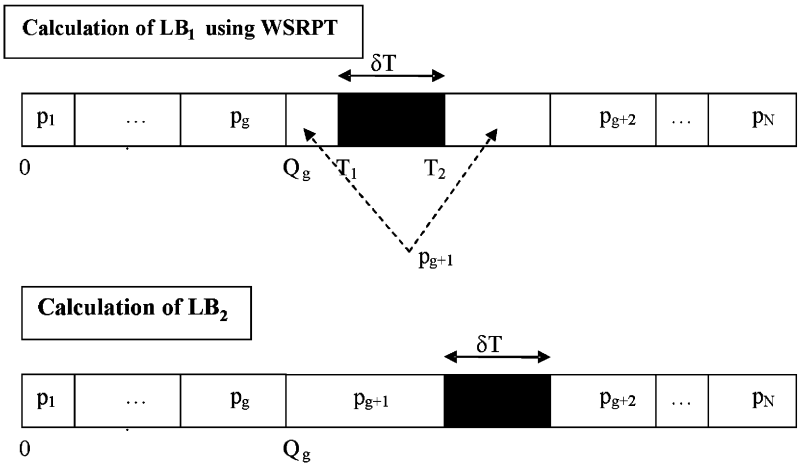


Fig. 7. Comparison of LB_1 and LB_2^*.

It should also be noted that

$$\lim_{w_t \to +\infty} LB_2(w_t) = -\infty \tag{9}$$

and

$$\lim_{w_t \to +\infty} \frac{LB_2(w_t)}{w_t} = -T_1. \tag{10}$$

Fig. 6 represents the curve produced for function LB_2 using the problem presented in Example 1. This confirms that $LB_2^* = LB_2((\delta T/p_{g+1}) \cdot w_{g+1})$.

Remark 1. LB_2^* can be computed in $O(N)$ if the jobs are already sorted according to the WSPT rule.

2.2.2. Comparison of LB_1 and LB_2^*

Fig. 7 represents the two bounds LB_1 and LB_2^*. LB_1 is calculated using the WSRPT rule according to the formula given in Proposition 2, and LB_2^* is calculated by scheduling jobs in the order obtained with WSPT and by introducing the dummy job t between $(g+1)$ and $(g+2)$ with $w_t = (\delta T/p_{g+1}) \cdot w_{g+1}$.

Fig. 8. The split used in lower bound LB_3.

From Eqs. (3) and (8), it can be established that

$$LB_2^* - LB_1 = \frac{\delta T}{p_{g+1}} w_{g+1}(Q_{g+1} - T_1).$$

The term $(Q_{g+1} - T_1)$ is strictly positive, therefore the following lemma holds.

Lemma 3. $LB_2^* > LB_1$.

2.3. Lower bounds based on splitting

In this subsection, we propose two other lower bounds based on the splitting technique introduced by Belouadah et al. [20] for the problem $1|r_i|\sum w_i C_i$. This technique has been used by other authors to solve diverse problems (For an example involving parallel machines, see [21].).

2.3.1. Splitting principle

The splitting principle was introduced by Belouadah et al. [20]. According to this principle, jobs are decomposed into pieces, with each job i being split into n_i pieces. Each piece (i, k) has a processing time p_i^k and a weight w_i^k ($1 \leqslant i \leqslant N$ and $1 \leqslant k \leqslant n_i$), such that $p_i = \sum_{k=1}^{n_i} p_i^k$ and $w_i = \sum_{k=1}^{n_i} w_i^k$. The problem (P) becomes scheduling the pieces such that, for each job i, the pieces (i, k) are scheduled contiguously. This new problem is noted (P').

Let C_i denote the completion time of job i in an optimal schedule of P, and C_i^k, the completion time of piece (i, k) in the corresponding optimal schedule for P'. Belouadah et al. proposed the following relation:

$$w_i C_i = \sum_{k=1}^{n_i} w_i^k C_i^k + \sum_{k=1}^{n_i-1} w_i^k \left(\sum_{l=k+1}^{n_i} p_i^l \right).$$

Therefore, the optimal value F^* can be written as follows:

$$F^* = \sum_{i=1}^{N} w_i C_i = \sum_{i=1}^{N} \sum_{k=1}^{n_i} w_i^k C_i^k + \sum_{i=1}^{N} \sum_{k=1}^{n_i-1} w_i^k \left(\sum_{l=k+1}^{n_i} p_i^l \right). \tag{11}$$

The quantity $(\sum_{i=1}^{N} \sum_{k=1}^{n_i} w_i^k C_i^k)$ represents the weighted sum of the completion times of the various job pieces. Therefore, $F^*(P) = F^*(P') + \sum_{i=1}^{N} \sum_{k=1}^{n_i-1} w_i^k (\sum_{l=k+1}^{n_i} p_i^l)$.

By relaxing the contiguity constraint in problem P', we obtain a relaxed problem denoted as (RP'), such that

$$F^*(P) \geqslant F^*(RP') + \sum_{i=1}^{N} \sum_{k=1}^{n_i-1} w_i^k \left(\sum_{l=k+1}^{n_i} p_i^l \right). \tag{12}$$

2.3.2. Lower bound LB_3

Relation (12) holds true for any split. Consider a special split: $\forall i \neq g+1,\ n_i = 1$ (Fig. 8). (These jobs are not actually split.)

Given $i = g + 1$, $n_i = 2$ is used, such that

- $p_{g+1}^1 = \delta,\ p_{g+1}^2 = p_{g+1} - \delta.$
- $w_{g+1}^1 = w_{g+1} \cdot \delta / p_{g+1}.$

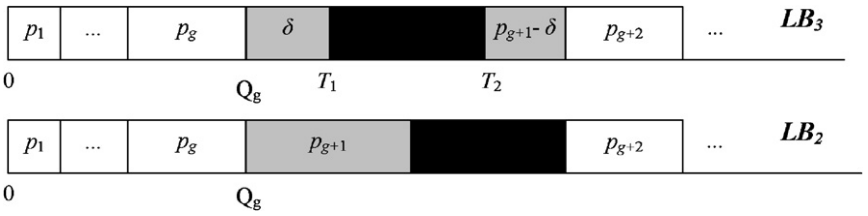


Fig. 9. Comparison of LB_2^* and LB_3.

- $w_{g+1}^2 = w_{g+1} \cdot (1 - \delta/p_{g+1}) = w_{g+1}(Q_{g+1} - T_1)/p_{g+1}$.

Thus, according to Lemma 1, the problem (RP') associated to this split can be solved optimally using the WSPT rule, thus making $LB_3 = F^*(RP') + w_{g+1}^1 \cdot (p_{g+1} - \delta)$ a valid lower bound (Fig. 9).

Theorem 1. $LB_3 = LB_2^*$.

Proof. By definition

$$\begin{aligned} LB_3 &= \sum_{i=1}^{g} w_i Q_i + w_{g+1}^1(Q_g + \delta) + w_{g+1}^2(Q_{g+1} + \delta T) + \sum_{i=g+2}^{N} w_i(Q_i + \delta T) + w_{g+1}^1(p_{g+1} - \delta) \\ &= \sum_{i=1}^{g} w_i Q_i + w_{g+1}^1 Q_{g+1} + w_{g+1}^2(Q_{g+1} + \delta T) + \sum_{i=g+2}^{N} w_i(Q_i + \delta T) \\ &= \sum_{i=1}^{g+1} w_i Q_i + w_{g+1}^2 \delta T + \sum_{i=g+2}^{N} w_i(Q_i + \delta T) \\ &= \sum_{i=1}^{g+1} w_i Q_i + w_{g+1} \frac{Q_{g+1} - T_1}{p_{g+1}} \delta T + \sum_{i=g+2}^{N} w_i(Q_i + \delta T) \end{aligned}$$

which is exactly the same expression as Eq. (8).

2.3.3. Lower bound LB_4

The lower bound LB_4 is based on the following split:

$$\forall i \leqslant N, \quad n_i = w_i \quad \text{and} \quad \forall 1 \leqslant k \leqslant n_i, \quad p_i^k = \frac{p_i}{w_i}, \quad w_i^k = \frac{w_i}{w_i} = 1.$$

Given this split, the relaxed problem (RP') is reduced to a problem of scheduling $(\sum_i w_i)$ jobs so as to minimize the total (unweighted) completion times.

Applying the SRPT[3] rule produces a lower bound LB_4 for the problem and produces the following sequence:

$$p_1^1, p_1^2, \ldots, p_1^{w_1}, p_2^1, p_2^2, \ldots, p_2^{w_2}, \ldots, p_N^1, p_N^2, \ldots, p_N^{w_N}$$

because $p_i^k = p_i/w_i$ and $p_1/w_1 \leqslant p_2/w_2 \leqslant \cdots \leqslant p_N/w_N$.

[3] SRPT: Shortest remaining processing time.

It is easy to verify that, in this sequence, the pieces of job i, such that $i \neq g+1$, are scheduled contiguously. Consequently,

$$\sum_{k=1}^{n_i} w_i^k C_i^k + \sum_{k=1}^{n_i-1} w_i^k \left(\sum_{l=k+1}^{n_i} p_i^l \right) = \begin{cases} w_i Q_i & \text{if } i < g+1, \\ w_i (Q_i + \delta T) & \text{if } i > g+1. \end{cases}$$

Let

$$h = \left\lfloor \delta \cdot \frac{w_{g+1}}{p_{g+1}} \right\rfloor.$$

The h first pieces of job $g+1$ can be completed before T_1. The possibility exists, however, that the processing of the $(h+1)$th piece will be interrupted and will resume again after T_2. This can be expressed as

$$C_{g+1}^k = \begin{cases} Q_g + k \dfrac{p_{g+1}}{n_{g+1}} & \text{if } k \leqslant h, \\ Q_g + k \dfrac{p_{g+1}}{n_{g+1}} + \delta T & \text{otherwise} \end{cases}$$

and, therefore,

$$\begin{aligned} \sum_{k=1}^{n_{g+1}} w_{g+1}^k C_{g+1}^k + \sum_{k=1}^{n_{g+1}-1} w_{g+1}^k \left(\sum_{l=k+1}^{n_{g+1}} p_{g+1}^l \right) &= \sum_{k=1}^{h} \left(Q_g + k \frac{p_{g+1}}{n_{g+1}} \right) + \sum_{k=h+1}^{n_{g+1}} \left(Q_g + k \frac{p_{g+1}}{n_{g+1}} + \delta T \right) \\ &\quad + \sum_{k=1}^{n_{g+1}-1} \left(\sum_{l=k+1}^{n_{g+1}} \frac{p_{g+1}}{n_{g+1}} \right) \\ &= n_{g+1} Q_g + n_{g+1} p_{g+1} + (n_{g+1} - h)\delta T \\ &= w_{g+1} Q_{g+1} + (w_{g+1} - h)\delta T. \end{aligned}$$

As a result,

$$LB_4 = \sum_{i=1}^{g+1} w_i Q_i + \sum_{i=g+2}^{N} w_i (Q_i + \delta T) + (w_{g+1} - h)\delta T.$$

By construction, the following theorem therefore holds.

Theorem 2. $LB_4 - LB_3 = \delta T((\delta / p_{g+1}) \cdot w_{g+1} - h) \geqslant 0$.

Proof. Comparing LB_4 with LB_3 yields

$$\begin{aligned} LB_4 - LB_3 &= \delta T (w_{g+1} - h) - \delta T \frac{w_{g+1}}{p_{g+1}} (Q_{g+1} - T_1) \\ &= \delta T \left(w_{g+1} - h - \frac{w_{g+1}}{p_{g+1}} (Q_{g+1} - T_1) \right) \\ &= \delta T \left(\frac{\delta}{p_{g+1}} \cdot w_{g+1} - h \right) \geqslant 0. \end{aligned}$$

Therefore, the result holds.

Furthermore, since LB_3 can be calculated in $O(N)$ time, LB_4 can also be calculated in $O(N)$ time by adding to LB_3 a constant term that can be implemented in $O(1)$ time.

3. Three exact approaches

In this section, we propose three exact approaches for solving the problem introduced at the beginning of this paper. The first approach uses a BAB algorithm based on the results presented in Section 2. The second approach applies

a mixed integer programming model. The last method exploits dynamic programming. Each of the approaches is described below.

3.1. Branch-and-bound (BAB) method

Based on the new lower bound LB_4, we propose a BAB method for solving the scheduling problem. The jobs are sorted according to the WSPT rule. Obviously, in the optimal solution, the set of jobs scheduled before T_1 are sorted according to the WSPT rule. The same is true for the jobs scheduled after T_2. Therefore, in our BAB method, a node is characterized by the following elements:

- a level k representing the number of scheduled jobs,
- a partial assignment of the scheduled jobs: $A = \{x_1, x_2, \ldots, x_k\}$ with $x_i \in \{0, 1\}$ $\forall i \leqslant k$ with $x_i = 1$ if job i is scheduled before T_1 and $x_i = 0$ if job i is scheduled after T_2, and
- a lower bound based on the results reported in Section 2.

The upper bound is computed using the heuristics, WSPT and MWSPT[4] [12]. When branching from a level k node, the position of job $(k + 1)$ must be chosen. It is either be scheduled before T_1 (in this case, $x_{k+1} = 1$) or after T_2 ($x_{k+1} = 0$). Once the lower bound has been calculated, the node is removed from the active list if the lower bound is greater than the current upper bound.

3.2. Mixed integer programming model

In this paragraph, we propose a mixed integer programming formulation. Such a model can be described as follows:

$$\text{Minimize} \quad \sum_{i=1}^{N} w_i \cdot C_i$$

Subject to

$$C_i \geqslant (1 - x_i) \cdot \left(T_2 + \sum_{j=1}^{i} p_j \right) - \sum_{j=1}^{i} x_j \cdot p_j \quad \forall 1 \leqslant i \leqslant N,$$

$$C_i \geqslant \sum_{j=1}^{i} x_j \cdot p_j \quad \forall 1 \leqslant i \leqslant N,$$

$$\sum_{j=1}^{N} x_j \cdot p_j \leqslant T_1,$$

$$x_i \in \{0, 1\} \quad \forall 1 \leqslant i \leqslant N$$

such that $x_i = 1$ if job i is scheduled before T_1 and $x_i = 0$ if job i is scheduled after T_2. C_i denotes the completion time for job i.

3.3. Dynamic programming

The problem is to partition jobs into two subsets Ω_1 and Ω_2, such that jobs in subset Ω_1 are scheduled before T_1 and jobs in subset Ω_2 are scheduled after. Due to the dominance of the WSPT rule, the jobs in each subset are scheduled in non-decreasing order according to their index.

Let $f(i, t)$ be the smallest total weighted completion time for a solution to the set of jobs $\{1, 2, \ldots, i\}$, where the total processing time of jobs in Ω_1 is t. If such a solution does not exist, we set $f(i, t) = +\infty$. The dynamic formulation

[4] The second heuristic applies the WSPT rule and inserts jobs that are already scheduled after T_2, before T_1. To this end, we try to insert the jobs scheduled after T_2 according to the WSPT rule. The procedure is stopped when no job can be inserted.

of the problem is based on the following recursive two-term equation, in which the first term represents the value of $f(i,t)$ if job i is scheduled after T_2, and the second term represents the value $f(i,t)$ if job i is scheduled before T_1

$$f(i,t) = \min \begin{cases} f(i-1,t) + w_i \cdot \left(T_2 + \sum_{k=1}^{i} p_k - t \right), \\ f(i-1, t-p_i) + w_i \cdot t, \end{cases} \tag{13}$$

where $f(0,0) = 0$, $f(0,t) = +\infty \ \forall t \leqslant T_1$ and $F^* = \min_{0 \leqslant t \leqslant T_1} f(N,t)$.

The time complexity of this algorithm is $O(N \cdot T_1)$.

4. Experimental results

In this section, we provide the computational results used to evaluate the performance of the different methods presented above. The tests were carried out on an AMD Athlon PC in the Windows XP environment. The following paragraphs describe our data generation methods, the results obtained and our analysis of these experiments.

4.1. Data generation

Three series of instance were randomly generated, such that $p_i \in [1, 100]$ and $w_i \in [1, 15]$ according to a discrete uniform distribution. To evaluate the impact of the timing of the unavailability period on the performance of the algorithms, T_1 and T_2 were generated as follows:

First series: $T_1 = \frac{1}{2} \cdot \sum_i p_i$ and $T_2 = T_1 + (1/N) \cdot \sum_i p_i$.

Second series: $T_1 = \frac{1}{4} \cdot \sum_i p_i$ and $T_2 = T_1 + (1/N) \cdot \sum_i p_i$.

Third series: $T_1 = \frac{3}{4} \cdot \sum_i p_i$ and $T_2 = T_1 + (1/N) \cdot \sum_i p_i$.

4.2. Results

The BAB method and the dynamic programming method were coded in the C language and were tested on the three series of problems. The mixed integer program was solved using CPLEX software on the same series of instances.

The different results are summarized in Tables 1–3. For each method, the computation time required to find the best solution is given. The "*best of*" column indicates the best solution found before stopping the program. The program is stopped if the optimum is found or if a CPU time limit was reached. For the BAB method and the mixed integer programming model, the number of nodes generated is also indicated. The symbol "$*$" indicates that the program was stopped before the optimal solution was found.

Table 1
Computational results for the mixed integer programming model

N	Series 1			Series 2			Series 3		
	Best of	Time	Nodes	Best of	Time	Nodes	Best of	Time	Nodes
5	3841	0.01	26	4177	0.01	0	3351	0.01	0
10	17 644	0.04	13	18 473	0.03	8	16 601	0.04	22
15	35 597	0.07	60	37 914	0.08	50	35 115	0.09	72
20	64 066	0.33	343	65 970	0.16	132	62 622	0.31	322
25	96 615	1.72	1639	99 710	0.6	686	94 486	0.89	970
30	144 238	7.33	6939	147 379	1.82	1863	141 833	3.41	2976
35	162 082	25.8	17 588	165 473	3.24	2885	166 016	6.58	5334
40	209 672	97.7	65 798	212 766	12.16	9587	206 330	21.8	16 074
45	270 505	452.4	303 680	274 927	54.2	39 428	267 351	90.31	57 374
50	334 288	2977	1 387 383	339 344	134.5	81 532	330 862	661.1	366 441
60	460 778*	1927	816 171	465 342	826.4	375 481	456 003	14 928	6 972 634
80	787 270*	11 171	2 989 682	794 254*	7655	2 431 887	783 474*	11 053	3 454 997

Table 2
Computational results for the branch-and-bound method

N	Series 1			Series 2			Series 3		
	Best of	Time	Nodes	Best of	Time	Nodes	Best of	Time	Nodes
5	3841	0	6	4177	0	10	3351	0	5
10	17 644	0	22	18 473	0	17	16 601	0	13
15	35 597	0	16	37 914	0	66	35 115	0	20
20	64 066	0	25	65 970	0	78	62 622	0	35
25	96 615	0	65	99 710	0	230	94 486	0	37
30	144 238	0	97	147 379	0	37	141 833	0	43
35	162 082	0	63	165 473	0	122	166 016	0	66
40	209 672	0	279	212 766	0	89	206 330	0	40
45	270 505	0	251	274 927	0.01	431	267 351	0	86
50	334 288	0	283	339 344	0.01	311	330 862	0	79
60	459 607	0	179	465 342	0	89	456 003	0.01	652
80	786 214	0.01	591	793 384	0.01	529	780 699	0.02	809
100	1 197 001	0.03	1034	1 205 941	0.01	456	1 190 758	0.02	842
150	2 533 074	0.05	1641	2 546 186	0.02	818	2 524 283	0.05	1394
200	4 288 875	0.08	1835	4 306 042	0.17	3941	4 277 766	0.03	894
250	6 272 140	0.32	6215	6 292 505	0.04	871	6 258 978	0.41	7774
300	9 014 746	0.27	4383	9 039 334	0.18	2838	8 998 975	0.05	916
350	12 354 413	0.14	2086	12 383 316	0.05	813	12 335 896	0.04	641
400	16 149 606	0.37	4352	16 183 102	1.97	23 555	16 128 628	0.78	9441
450	20 093 503	24.2	30 010	20 130 749	60.4	52 545	20 069 866	0.42	4719
500	24 513 907	0.64	6339	24 554 984	59.5	48 044	24 488 413	1.89	18 572
550	29 730 531	4.51	17 393	29 776 004	0.74	6819	29 702 046	0.38	3673
600	36 168 484*	292	54 950	36 218 545	0.73	6194	36 136 698	1.79	14 776
650	43 443 480*	231	50 726	43 498 188	10.8	22 505	43 407 996	0.79	6081
700	51 053 065*	214	47 860	51 113 422	121	38 117	51 014 387*	198	47 694
750	58 998 439*	262	43 715	59 062 841	1.02	7026	58 956 919*	1377	44 471
800	66 838 958*	240	40 966	6 690 819	0.32	2192	66 795 264*	248	41 691
850	75 309 474	9.32	10 287	75 382 531	0.26	1746	75 262 990*	197	39 293
900	82 991 955*	467	36 866	83 058 702	0.26	1588	82 943 207*	197	37 216
950	91 283 109	26.8	19 828	91 364 090	1.71	9561	91 233 621*	219	35 080
1000	101 620 801*	327	33 134	101 705 362*	287	33 116	101 565 993*	188	33 631

According to these results, it is obvious that the BAB algorithm and the dynamic programming algorithm are much more effective than the mixed integer programming model. The time needed for the BAB and the dynamic programming algorithm is usually less than for the mixed integer programming method (Fig. 10). The latter approach was not able to solve problems with more than 60 jobs. The BAB method, on the other hand, was able to solve problems with close to 1000 jobs. However, the results obtained show that the dynamic programming method clearly outperformed the other two. It was able to solve instances in the second series with up to 3000 jobs, usually with a very short computation time and no problems due to memory requirement. The performance of these methods are summarized in Table 4 in which we report the maximum size of solved problems for each method and for each series.

4.3. Comparison of LB_4 with the linear bound

In an attempt to explain the large performance difference between the BAB algorithm and the mixed integer programming model, we compared the performance of the lower bound LB_4 used in our BAB algorithm and the linear bound, LB_{LP}, obtained by relaxing the integrity constraint in the mixed integer programming model. These lower bounds were computed for the three series of instances mentioned above. The results presented in Table 5 explain the effectiveness of our BAB algorithm. Indeed, the value of the lower bound LB_4 is usually close to the optimal solution value for all the instances, whereas the linear bound is usually less than LB_4 value, leading to a large number of nodes and long

Table 3
Computational results for the dynamic programming method

Series 1			Series 2			Series 3		
N	Best of	Time	*N*	Best of	Time	*N*	Best of	Time
5	3841	0	5	4177	0	5	3351	0
10	17 644	0	10	18 473	0	10	16 601	0
15	35 597	0	15	37 914	0	15	35 115	0.01
20	64 066	0.01	20	65 970	0.01	20	62 622	0.01
25	96 615	0.01	25	99 710	0	25	94 486	0.01
30	144 238	0.02	30	147 379	0.01	30	141 833	0.01
35	162 082	0.02	35	165 473	0	35	166 016	0.02
40	209 672	0.02	40	212 766	0.01	40	206 330	0.03
45	270 505	0.03	45	274 927	0.02	45	267 351	0.04
50	334 288	0.04	50	339 344	0.02	50	330 862	0.06
60	459 607	0.06	60	465 342	0.03	60	456 003	0.09
80	786 214	0.17	80	793 384	0.10	80	780 699	0.25
100	1 197 001	0.30	100	1 205 941	0.17	100	1 190 758	0.45
150	2 533 074	0.76	150	2 546 186	0.42	150	2 524 283	1.1
200	4 288 875	1.54	200	4 306 042	0.80	200	4 277 766	2.3
250	6 272 140	2.69	250	6 292 505	1.4	250	6 258 978	4
300	9 014 746	4.43	300	9 039 334	2.26	300	8 998 975	6.6
350	12 354 413	6.70	350	12 383 316	3.47	350	12 335 896	10
400	16 149 606	9.71	400	16 183 102	5.01	400	16 128 628	14
450	20 093 503	13.5	450	20 130 749	6.97	450	20 069 866	20
500	24 513 907	18.2	500	24 554 984	9.35	500	24 488 413	27
550	29 730 531	24.1	550	29 776 004	12.3	550	29 702 046	36
600	36 168 480	31.4	600	36 218 545	16.1	600	36 136 698	47
650	43 443 438	41.1	650	43 498 188	21.1	650	43 407 996	62
700	51 053 065	50.4	700	51 113 422	26.1	700	51 014 341	76
750	58 998 430	63.6	750	59 062 841	32.4	750	58 956 861	95
800	66 838 957	76.4	800	6 690 819	39.1	800	66 795 002	115
850	75 309 474	88.3	850	75 382 531	44.6	850	75 262 943	132
900	82 991 955	106	900	83 058 702	54.1	900	82 943 203	159
950	91 283 109	119	950	91 364 090	61.2	950	91 232 667	180
1000	101 619 985	142	1000	101 705 085	72.8	1000	101 565 993	214
1100	123 113 699	193	1200	146 061 852	131	1100	123 053 874	286
1200	145 949 346	261	1400	198 743 126	204	1200	145 883 922	384
1400	198 624 074	406	1500	228 675 168	282	1300	172 869 143	494
1500	228 545 688	564	2000	407 772 898	659	1400	198 549 529	609
2000	407 598 503	1280	2500	632 524 601	1403	1500	228 463 861	838
2500	632 308 500	2741	3000	894 959 660	2338	2000	407 489 833	2398

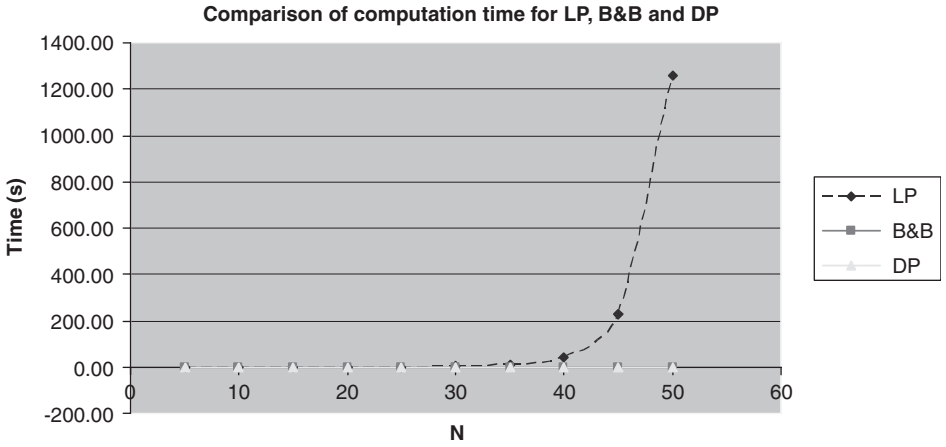


Fig. 10. Computation time for the various algorithms.

Table 4
The maximum size of solved problems

	ILP	BAB	DP
Series 1	50	550	2500
Series 2	60	950	3000
Series 3	60	650	2000

Table 5
Comparison of lower bounds LB_4 and LB_{LP}

N	LB_4	LB_{LP}	Best of	Ratio LB_4	Ratio LB_{LP}
5	3109	2305	3790	82.04	60.82
10	17 046	12 072	17 573	97.00	68.70
15	35 570	14 778	36 209	98.24	68.43
20	63 767	44 140	64 219	99.30	68.73
25	96 281	66 593	96 937	99.32	68.70
30	144 211	99 594	144 483	99.81	68.93
35	162 110	112 094	164 524	98.53	68.13
40	209 285	144 559	209 589	99.85	68.97
45	270 419	186 519	270 928	99.81	68.84
50	334 670	230 246	334 831	99.95	68.76
60	460 016	316 275	460 317	99.93	68.71
80	786 536	541 400	786 855	99.96	68.81
100	1 197 629	824 897	1 197 900	99.98	68.86
150	2 534 244	1 743 298	2 534 514	99.99	68.78
200	4 290 689	2 949 166	4 290 894	100.00	68.73
250	6 274 384	4 313 966	6 274 541	100.00	68.75
300	9 017 557	6 200 505	9 017 648	100.00	68.76
350	12 357 828	8 503 942	12 357 875	100.00	68.81
400	16 153 624	1 114 307	16 153 779	100.00	68.80
450	20 097 844	14 736 420	20 098 039	100.00	73.32
500	24 518 929	15 965 743	24 519 101	100.00	65.12
550	29 736 112	20 458 165	29 736 194	100.00	68.80
600	36 174 410	24 881 107	36 174 576	100.00	68.78
650	43 449 724	29 882 155	43 449 888	100.00	68.77
700	51 060 074	35 116 882	51 060 291	100.00	68.78
750	59 005 825	40 574 432	59 006 066	100.00	68.76
800	66 847 176	45 971 541	66 847 414	100.00	68.77
850	75 318 234	51 798 269	75 318 332	100.00	68.77
900	83 001 170	57 089 725	83 001 288	100.00	68.78
950	91 293 104	62 791 292	91 293 607	100.00	68.78
1000	101 630 134	69 896 254	101 630 719	100.00	68.77

computation times (see Tables 1 and 2). Fig. 11 shows the ratio ($100\% \cdot \textit{lower bound}/OPT$) obtained for each lower bound (LB_4 and LB_{LP}). This ratio clearly shows that LB_4 is much tighter than LB_{LP}.

4.4. Comparison of the BAB and the dynamic programming methods

From the results presented in Tables 2 and 3, we can conclude that the dynamic programming method outperforms the BAB for the different simulations and is able to solve problems of 3000 jobs. In these cases, the complexity of dynamic programming remains reasonable. However, for the instances with more than 3000 jobs, the dynamic programming method becomes unsuitable because the term T_1 in the complexity formula becomes very high. This term becomes very large when $\sum_{1 \leqslant i \leqslant N} p_i$ is large. Thus, for very high values of T_1, the BAB method would appear to be more suitable for solving the problem or to find a near optimal solution. Please note that, for the second series, the term T_1

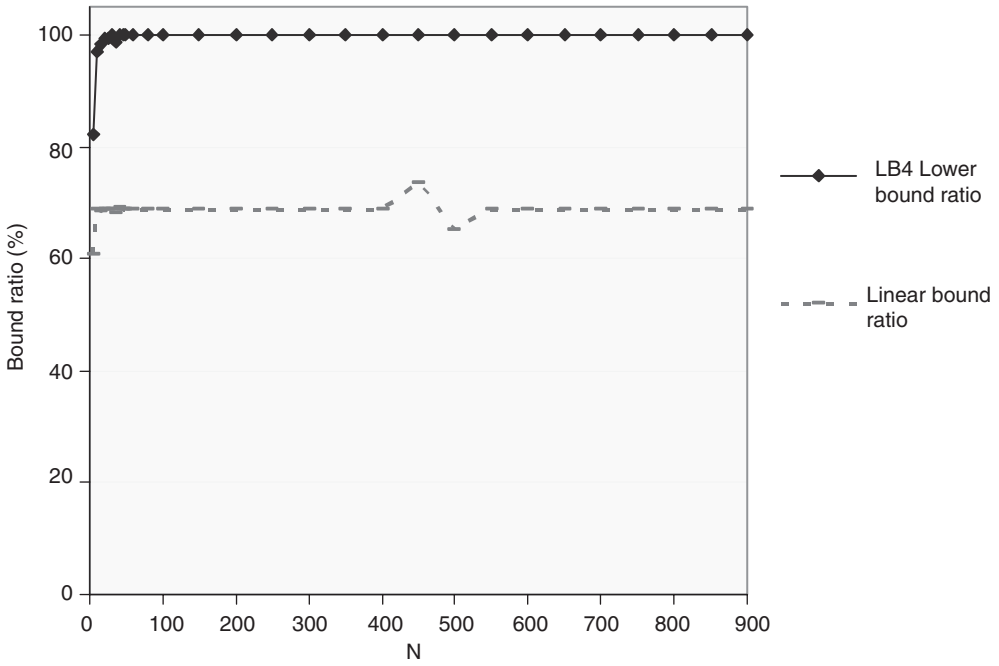


Fig. 11. Comparison of LB_4 and LB_{LP}.

is not very large ($T_1 = \frac{1}{4} \cdot \sum_{1 \leqslant i \leqslant N} p_i$) and thus the dynamic programming method is very efficient at solving the problems.

Given the above observations, we conclude that the dynamic programming method is unsuitable for solving problems with a high T_1 value. This is the case when maintenance is scheduled far in the future or when many jobs can be scheduled in the set before the beginning of maintenance. To confirm this conclusion, we generated a fourth series of instances as follows:

- $N \in \{500, 600, 700, 800, 900, 1000, 1200\}$,
- $p_i \in [1, 250]$ according to a discrete uniform distribution,
- $w_i \in [1, 50]$ according to a discrete uniform distribution,
- $T_1 = \frac{1}{2} \cdot \sum_i p_i$,
- $T_2 = T_1 + (1/N) \cdot \sum_i p_i$.

The different results obtained for this series are reported in Table 6 and graphically illustrated in Fig. 12. Clearly, the dynamic programming is less efficient than the BAB for these instances. The higher computation time required by the dynamic programming algorithm confirms our conclusions.

4.5. Impact of the interval length

In the experiment presented here, we assumed that the unavailability interval length is equal to the mean processing time. However, because this variable can influence the results, it is interesting to analyze this effect to understand the behavior of the methods presented above.

For the dynamic programming method, the variation of the interval length has no effect on the performance of this method. Indeed, the dynamic programming algorithm is implemented in $O(N \cdot T_1)$ time and therefore, its complexity is independent of the interval length.

Additional numerical experiments showed that it is very hard to quantify the possible effect on the performances of the BAB method and the mixed integer programming method. Indeed, increasing the interval length increases the

Table 6
Computational results for the branch and bound and the dynamic programming: fourth series

N	OPT	$Time_{B\&B}$	$Time_{DP}$
500	199 075 673	0.79	56.61
600	293 949 624	17.68	96.81
700	414 698 746	52.24	176.33
800	542 934 053	33	232.69
900	673 722 106	44.6	322.31
1000	823 782 052	6.47	449.31
1200	1 185 505 044	35.76	821.05

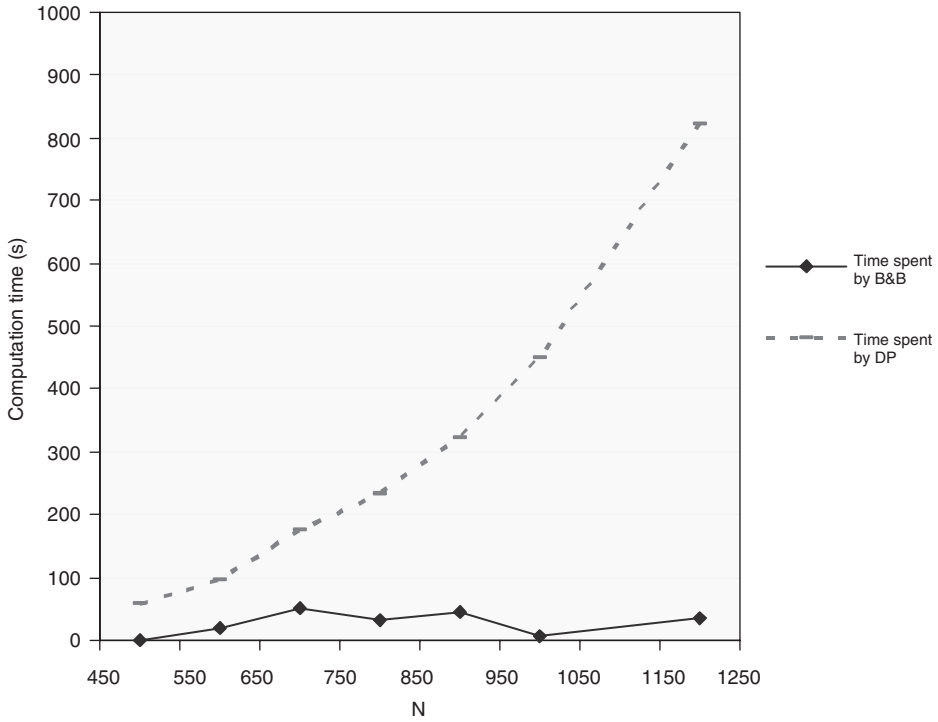


Fig. 12. Computation times for the branch-and-bound method and the dynamic programming method: fourth series.

lower bounds LB_4 and LB_{LP}, but the optimal value of the sum of completion times increases too. Therefore, we cannot determine whether the problem will become easier or harder.

Fig. 13 summarizes the results of three tests illustrating this difficulty for the BAB method. Each test represents a comparison of the program success ratio obtained for a short interval length ($\delta T = (1/N) \cdot \sum_i p_i$) and a large interval length ($\delta T = (5/N) \cdot \sum_i p_i$). The success ratio is defined as the percentage of problems solved to optimality in each series. Since we are interested in evaluating the performance of the method for a large problem size, we tested three additional random series ($N = 1000; 950; 800$). As Fig. 13 clearly shows the effect of the length interval is very hard to characterize analytically. Statistically, we can only conclude that mean performance is not sensitive to variations in interval length.

5. Conclusion

In this paper, we consider the scheduling problem for a single machine with an availability constraint to minimize the weighted sum of completion times. We proposed several new lower bounds and studied their mathematical properties.

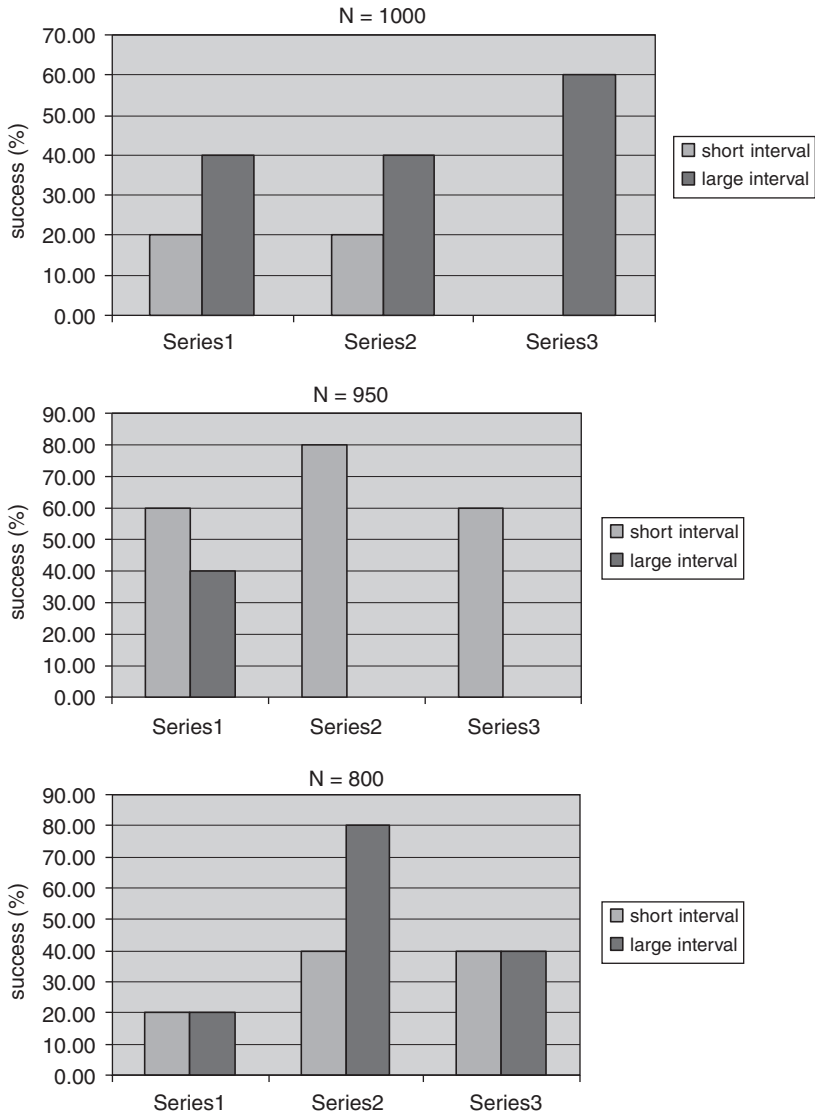


Fig. 13. Effect of the interval length variation on BAB performance.

We then proposed a BAB method based on one of these new lower bounds, the LB_4. This BAB method was tested against a method using mixed integer programming and one using dynamic programming. The results show that the BAB method and the dynamic programming method performed better than the integer programming method, and the complementarity of the BAB and the dynamic programming methods. In our future research, we hope to extend these methods to others variants of this problem.

References

[1] Adiri I, Bruno J, Frostig E, Rinnooy Kan AHG. Single machine flow-time scheduling with a single breakdown. Acta Informatica 1989;26: 679–96.

[2] Lee C-Y, Liman SD. Single machine flow-time scheduling with scheduled maintenance. Acta Informatica 1992;29:375–82.

[3] Sadfi C, Penz B, Rapine C, Blażewicz J, Formanowicz P. An improved approximation algorithm for the single machine total completion time scheduling problem with availability constraints. European Journal of Operational Research 2005;161:3–10.

[4] Sadfi C, Penz B, Rapine C. A dynamic programming algorithm for the single machine total completion time scheduling problem with availability constraints. Eighth international workshop on project management and scheduling, Valence, Spain, 2002.

[5] Qi X, Chen T, Tu F. Scheduling the maintenance on a single machine. Journal of the Operational Research Society 1999;50:1071–8.
[6] Chen WJ. Minimizing total flow time in the single-machine scheduling problem with periodic maintenance. Journal of the Operational Research Society 2005; 1–6.
[7] Lee C-Y, Liman SD. Capacitated two-parallel machines scheduling to minimize sum of job completion times. Discrete Applied Mathematics 1993;41:211–22.
[8] Mosheiov G. Minimizing the sum of job completion times on capacitated parallel machines. Mathematical and Computer Modelling 1994;20: 91–9.
[9] Lee C-Y. Machine scheduling with an availability constraints. Journal of Global Optimization 1996;9:363–84.
[10] Sadfi C, Ouarda Y. Parallel machines scheduling problem with availability constraints. Ninth international workshop on project management and scheduling, Nancy, France, 2004.
[11] Wang G, Sun H, Chu C. Preemptive scheduling with availability constraints to minimize total weighted completion times. Annals of Operations Research 2005;133:183–82.
[12] Kacem I, Chu C. Worst case performance analysis of WSPT and MWSPT for 1, $h_1 \| \sum w_i \cdot C_i$. Proceedings of IESM, Marrakech, Maroco, 2005.
[13] Kacem I, Chu C, Souissi A. Minimizing the weighted sum of completion times with availability constraint: comparison of branch and bound method and integer linear programming. Proceeding of IESM, Marrakech, Maroco, 2005.
[14] Aggoune R. Minimizing the makespan for the flow shop scheduling problem with availability constraints. European Journal of Operational Research 2004;153:534–43.
[15] Aggoune R, Portmann M-C. Flow shop scheduling problem with limited machine availability: a heuristic approach. International Journal of Production Economics 2006;99:4–15.
[16] Sanlaville E, Schmidt G. Machine scheduling with availability constraints. Acta Informatica 1998;35:795–811.
[17] Lee C-Y. Machine scheduling with availability constraints. In: Leung JY-T, editor. Handbook of scheduling: algorithms, models, and performance analysis. Boca Raton, FL, USA: CRC Press; 2004.
[18] Türkcan A. Machine scheduling with availability constraints. Report available at ⟨ayten@bilkent.edu.tr⟩; 1999.
[19] Lee C-Y. Machine scheduling with an availability constraint. Journal of Global Optimization 1996;9:363–82.
[20] Belouadah H, Posner ME, Potts CN. Scheduling with release dates on a single machine to minimize total weighted completion time. Discrete Applied Mathematics 1992;36:213–31.
[21] Webster S. Weighted flow time bounds for scheduling identical processors. European Journal of Operational Research 1995;80:103–11.
[24] Lee C-Y, Chen Z-L. Scheduling of jobs and maintenance activities on parallel machines. Naval Research Logistics 2000;47:145–65.