Minimizing the sum of weighted completion times with unrestricted weights

Mauro Dell'Amico*, Silvano Martellob,*, Daniele Vigo

a Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy
b Dipartimento di Elettronica, Informatica e Sistemistica, Università di Bologna,
Viale Risorgimento 2, 40136 Bologna, Italy

Received 28 May 1993; revised 4 February 1994

Abstract

Given a set of tasks with associated processing times, deadlines and weights unrestricted in sign, we consider the problem of determining a task schedule on one machine by minimizing the sum of weighted completion times. The problem is NP-hard in the strong sense. We present a lower bound based on task splitting, an approximation algorithm, and two exact approaches, one based on branch-and-bound and one on dynamic programming. An overall exact algorithm is obtained by combining these two approaches. Extensive computational experiments show the effectiveness of the proposed algorithm.

Keywords: Scheduling; Single machine; Earliness; Branch-and-bound; Dynamic programming

1. Introduction

Given a machine which can process at most one task at a time, and a set $T = \{T_1, \ldots, T_n\}$ of $n$ tasks with associated processing times $p_1, \ldots, p_n$, deadlines $d_1, \ldots, d_n$, and weights $w_1, \ldots, w_n$, we consider the problem (called $P$ in the following) of determining a task schedule that minimizes the sum of weighted completion times, while preserving the deadline requirements of each task. Using the three-field classification introduced in Graham et al. [7], the problem is denoted as $1\mid d\mid \sum w_j C_j$.

We assume that tasks are available at time zero, that processing times and deadlines are positive integers, and that weights are integers unrestricted in sign. A schedule is defined through the vector $(C_1, \ldots, C_n)$ of the task completion times: task $T_j$ is processed in time interval $(C_j - p_j, C_j]$. 

*Corresponding author.
The problem is strongly NP-hard, even if the weights are restricted to being positive integers (see Lenstra et al. [9]). An exact algorithm for this restricted case has been given by Posner [11].

The unrestricted case can be re-stated by expressing each weight, \( w_j \), as the difference between two nonnegative values \( \alpha_j \) (flow time penalty) and \( \beta_j \) (earliness penalty). The objective function of \( P \) can thus be written as

\[
Z^*(P) = \min \sum_{j=1}^{n} (\alpha_j C_j - \beta_j C_j)
\]

\[
= \min \sum_{j=1}^{n} (\alpha_j C_j + \beta_j (d_j - C_j)) - \sum_{j=1}^{n} \beta_j d_j
\]

Problem \( P' \) can be interpreted as a single machine scheduling problem where the processing cost associated with each task \( T_j \) is equal to \( \alpha_j \) times its completion time plus \( \beta_j \) times its earliness, \( E_j = d_j - C_j \). The flow time penalty has classically been used to model overhead and capital carrying costs sustained during production, while the earliness penalty takes into account the cost incurred for storing a finished product until it is shipped. Hence the model is useful to describe problems arising in a Just In Time context.

An exact algorithm for \( P' \), based on a dynamic programming approach, has been developed by Bard et al. [2], while Feo et al. [5] have presented a heuristic algorithm based on a Greedy Randomized Adaptive Search Procedure (GRASP). Special cases and related problems have also been studied by Fry and Leong [6], Bagchi and Ahmadi [1], Faaland and Schmitt [4] and Sen et al. [12].

In Section 2 we present a lower bound based on task splitting. In Section 3 we use the preemptive lower bound to obtain an approximation algorithm. Two exact algorithms, one based on branch-and-bound and one on dynamic programming, are developed in Section 4. Extensive computational experiments are presented in Section 5.

Throughout the paper we assume that the tasks are numbered so that

\[
d_1 \leq d_2 \leq \cdots \leq d_n.
\]

It is well known (see Smith [13]) that problem \( P \) has a feasible solution only if the schedule produced by the so-called earliest due date (EDD) rule \( (C_1 = p_1; C_j = C_{j-1} + p_j, j = 2, \ldots, n) \) is feasible.

2. Lower bound

Let us consider the following new problem, called \( SP \) in the sequel, obtained from \( P \) by allowing each task \( T_j \) to be split into any number \( k(j) \) of pieces \( T_{j,1}, \ldots, T_{j,k(j)} \) with
deadlines $d_{ij} = d_j$ for each $i$ and $j$, positive processing times $p_{j1}, \ldots, p_{jm_j}$ such that $\sum_{i=1}^{m_j} p_{ji} = p_j$ for each $j$, and weights $w_{j1}, \ldots, w_{j_{m_j}}$ with $w_{j1} = p_{j1} w_j / p_j$. Let $C_j$ be the completion time of piece $T_j$; the objective function of $SP$ is defined as

$$z(SP) = \sum_{j=1}^{n} \sum_{i=1}^{k(j)} w_{ji} C_{ji}. \quad (2)$$

Posner [11] proves (for the case $w_j \geq 0$, but it can be easily seen that the proof holds also for unrestricted $w_j$) that, given a feasible solution to $P$ of value $z(P)$, for any task splitting, the solution to $SP$ obtained by consecutively scheduling $T_{j1}, \ldots, T_{jk(j)}$, in time interval $(C_j - p_j, C_j]$ satisfies $z(P) = z(SP) + CBRK$, where

$$CBRK = \sum_{j=1}^{n} \sum_{i=1}^{k(j)-1} w_{ji} \sum_{h=i+1}^{k(j)} p_{jh}. \quad (3)$$

Let $z^*(SP)$ be the optimal solution value of $SP$; then

$$L = \lceil z^*(SP) + CBRK \rceil \quad (4)$$

is a valid lower bound on the value of $z(P)$. We show in the following that $SP$ can be solved in polynomial time.

Let us partition task set $T$ into $T^+ = \{ T_j : w_j \geq 0 \}$ and $T^- = \{ T_j : w_j < 0 \}$. Let $n^+$ and $n^-$ be the cardinalities of $T^+$ and $T^-$, respectively. These two subsets contain tasks that have a different behavior in an optimal schedule: the tasks of set $T^-$ require to be processed as late as possible, while those of set $T^+$ must be scheduled as early as possible. Let us also rename the tasks in such a way that

$$T^+ = \{ T^+_1, \ldots, T^+_{n^+} \}, \quad T^- = \{ T^-_1, \ldots, T^-_{n^-} \}$$

(with $d^+_{j1}, d^-_{j1}, p^+_{j1}, p^-_{j1}, w^+_{j1}, w^-_{j1}$ renamed accordingly) and that (1) holds for $T^+$ and $T^-$, i.e.,

$$d^+_{j1} \leq d^+_{j+1}, \quad d^-_{j1} \leq d^-_{j+1} \quad \text{for all } j. \quad (5)$$

**Definition 1.** A block is a set $B_i = \{ T^+_{a_i}, T^-_{a_i+1}, \ldots, T^-_{b_i} \}$ of consecutive tasks of $T^-$ (ordered according to (5)), whose total processing time is not greater than the time interval between $d^-_{a_i-1}$ (with $d^-_0 = 0$) and $d^-_{b_i}$. Let $s_i = d^-_{b_i} - \sum_{j=a_i}^{b_i} p^-_j$: the associated block interval is $BI_i = (s_i, d^-_{b_i}]$.

Arising from Definition 1 we have, for any task splitting: (a) no piece coming from a task of a block $B_i$ can be processed after the right extreme $d^-_{b_i}$; (b) all pieces coming from the tasks of $B_i$ can be processed in the associated block interval, leaving no idle time; (c) all the deadlines of these pieces, and no other deadline of a piece coming from a task of $T^-$, fall within the block interval.
Example 1. Let $n = 10$, $(p_j) = (2, 4, 3, 3, 3, 5, 4, 3, 5, 7)$, $(d_j) = (10, 10, 12, 29, 30, 32, 41, 43, 45, 46)$ and $(w_j) = (-1, 4, -4, 1, -3, 3, -5, -3, -1, 1)$. In Fig. 1(a) the tasks are right-justified on their deadlines (those with negative penalty drawn with heavy lines) and the ratios $w_j/p_j$ are given. The figure shows the three resulting block intervals.

Let us now define

$$
\tau = \min \left\{ t : \sum_{j=1}^{n} p_j^+ + \sum_{i:s_i < t} (d_{s_i}^- - s_i) = t \right\}
$$

(6)
and note that \( \tau \) cannot be the right extreme of any block interval. In the example it results \( \tau = 24 \). We can now divide problem \( SP \), i.e., the solution of \( P \) with task splitting allowed, into two subproblems:

-  \( P_A \): \( SP \) for the tasks in \( T_A = T^+ \cup \{ T_j : d_j < \tau \} \);
-  \( P_B \): \( SP \) for the tasks in \( T_B = \{ T_j : d_j > \tau \} \)

and observe that

\[
\sum_{j \in T_A} p_j = \tau. \tag{7}
\]

We will show that \( z^*(SP) \) can be determined by separately solving, in polynomial time, three subproblems induced by the tasks in \( T^+ \), \( T_A \setminus T^+ \) and \( T_B \), respectively.

**Theorem 1.** In the optimal solution to \( SP \) all pieces coming from tasks of \( T_A \) are scheduled in \((0, \tau]\).

**Proof.** Consider the optimal solution to \( SP \) and assume that some pieces coming from tasks of \( T_A \) are scheduled after \( \tau \). By definition of \( T_A \) all such pieces must come from tasks belonging to \( T^+ \): let \( T_{jk}^+ \) be, among these, the one with minimum completion time \( C_{jk} > \tau \). Observe that there is no idle time instant in \((0, C_{jk} - p_{jk}^+]\) since otherwise we could introduce a new split and move a unit of \( T_{jk} \) to such an instant, thus improving the solution. Hence, from (7), there is a set \( Q \) of pieces coming from tasks of \( T_B \), scheduled before \( C_{jk} - p_{jk}^+ \), with total processing time \( p_Q \geq C_{jk} - \tau \). From (6), all such pieces come from tasks whose blocks \( B_i \) satisfy \( s_i \geq \tau \), and we have already observed that all the tasks of a block can be scheduled within the associated block interval. Therefore at least one piece \( T_{jk}^+ \in Q \) must have deadline \( d_{jk} \geq C_{jk}^+ \), since \( \tau + p_Q \geq C_{jk} \). It follows that a unit of \( T_{jk}^+ \) and one of \( T_{jk}^- \) can be interchanged to improve the solution, which is a contradiction. 

**Corollary 1.** The separate optimal solutions to \( P_A \) and \( P_B \) do not overlap and they produce the optimal solution to \( SP \).

**Proof.** Immediate from Theorem 1 and Eq. (7).

**Theorem 2.** In the optimal solution to \( P_A \) any piece coming from a task \( T_j \in T_A \) is scheduled in the block interval associated with the block containing \( T_j \).

**Proof.** Assume the thesis is not true. Let \( BI_i \) be the rightmost block interval such that a piece coming from a task of \( B_i \) is not scheduled in \((s_i, d_{hi}^-]\), and observe that such a piece must be scheduled before \( BI_i \). Since in any optimal solution there is no idle time in \((0, \tau]\), at least one piece coming from a task of \( T^+ \) must be scheduled in \((s_i, d_{hi}^-]\) (indeed, no piece scheduled in \( BI_i \) could come from a task \( T_j^+ \in B_m, m \neq i \), through our choice of \( BI_i \) and the deadline constraints); let \( T_{jk}^+ \) denote the rightmost such piece. Let \( T_{jk}^- \) be the rightmost piece coming from a task of \( T^- \) and scheduled before \( C_{jk} \), with
Corollary 2. Problem $P_A$ decomposes into: (i) problem $P^-_A$ of optimally scheduling (with splitting) the tasks $T^-_i \in T_A$ in the corresponding block intervals; and (ii) problem $P^+_A$ of optimally scheduling (with splitting) the tasks of $T^+$ into intervals $(0, \tau] \backslash \{BI_i: s_i \leq \tau\}$.

We have thus shown that the optimal solution to $SP$ can be determined by separately solving:

1. $SP$ for the tasks in $T_B$ (problem $P_B$) in time interval $(\tau, +\infty)$;
2. $SP$ for the tasks in $T_A \backslash T^+$ (problem $P^-_A$) in time interval $(0, \tau]$, and then
3. $SP$ for the tasks in $T^+$ (problem $P^+_A$) in the intervals of $(0, \tau]$ not used for the solution of $P^-_A$.

In $P_B$ and $P^-_A$ all the task weights are negative. Any instance of these problems can then be transformed into an equivalent instance of $1|r_j|\Sigma v_j C_j$ (with splitting), obtained by setting $v_j = -w_j$ and $r_j = \max\{d_k\} - d_j$ for all $j$. This problem is exactly solved in $O(n \log n)$ time by the algorithm of Belouadah et al. [3].

In $P^+_A$ all the task weights are positive. A straightforward adaptation of the algorithm of Posner [11] (to take into account the forbidden intervals) exactly solves this problem in $O(n \log n)$ time.

We present a procedure which merges the above algorithms to solve all three problems at one time. In Step 1 we schedule tasks of $T^-$ within the associated block intervals. The schedules are determined from right to left, starting from time instant $\max_j\{d^-_j\}$. At any iteration, $t$ represents the maximum time instant which can be used for the completion of a new task and $E$ is the set of unscheduled tasks (and pieces) having deadline not less than $t$: from $E$ we select task $T_h$ with minimum value of the ratio $w_j/p_j$, and schedule it. If no deadline of a task with ratio $w_j/p_j < w_h/p_h$ falls in $(t - p_h, t]$, then $T_h$ is scheduled entirely and $t$ is set to $t - p_h$; otherwise it is partially scheduled by splitting it at the maximum, $d^*$, of such deadlines, and $t$ is set to $d^*$. Whenever $E = \emptyset$, $t$ jumps to the rightmost deadline of an unscheduled task of $T^-$ (i.e., to the right extreme of a new block interval). The execution of Step 1 terminates as soon as the total processing time, $s$, of the unscheduled tasks is greater than $t$ (so, from Theorem 1 and (7), we know that $\tau = s$); we then set $t = s$ and proceed with the solution of problem $P_A$.

Step 2 is very similar to Step 1, but the case $E = \emptyset$ can never occur (see (7)) and the next task to schedule is in $T^+$ (since $\tau$ cannot be the right extreme of any block interval). Step 3 computes a lower bound for $P$ according to (2)–(4). The pseudo-code follows.
Procedure LB

Step 0 (initialization)
\[ L := 0; \ CBRK := 0; \]
\[ s := \sum_{j=1}^{n} p_j; \ t := \max_j \{ d_j^- \} \quad (t := 0 \text{ if } T^- = \emptyset); \ E := \{ T_j^- : d_j^- = t\}; \ E^- := T^- \setminus E; \]

Step 1 (solution of problem \( P_B \))

while \( s \leq t \) do

\[ h := \arg \min \{ w_j/p_j : T_j \in E \}; \]
\[ F := \{ T_j \in E : d_j > t - p_h \}; \]

if \( \{ T_j \in F : w_j/p_j < w_h/p_h \} = \emptyset \) then

schedule \( T_h \) in \( (t - p_h, t] \); \( L := L + w_h t; \ s := s - p_h; \)
\[ t := t - p_h; \ F := F \cup \{ T_j \in E : d_j = t \}; \ E := (E \setminus \{ T_h \}) \cup F; \ E^- := E \setminus F; \]

if \( E = \emptyset \) then (comment: a block interval has been completed)

\[ t := \max \{ d_j : j \in E \} \quad (t := 0 \text{ if } E = \emptyset); \ E := \{ T_j : d_j = t \}; \ E^- := E \setminus E \]

endif

else call SPLIT

endwhile

Step 2 (solution of problems \( P_A \) and \( P_A' \))

\[ t := s \ (\text{comment: value of } \tau); \ E := \{ T_j^+ : d_j \geq t \}; \ E^- := \{ T_j : d_j < t \}; \]

while \( E \neq \emptyset \) do

\[ h := \arg \min \{ w_j/p_j : T_j \in E \}; \]
\[ F := \{ T_j \in E : d_j > t - p_h \}; \]

if \( \{ T_j \in F : w_j/p_j < w_h/p_h \} = \emptyset \) then

schedule \( T_h \) in \( (t - p_h, t] \); \( L := L + w_h t; \ s := s - p_h; \)
\[ t := t - p_h; \ F := F \cup \{ T_j \in E : d_j = t \}; \ E := (E \setminus \{ T_h \}) \cup F; \ E^- := E \setminus F; \]

else call SPLIT

endwhile

Step 3 (define the lower bound value)
\[ L := \lceil L + CBRK \rceil. \]

Procedure SPLIT

\[ d^* := \max \{ d_j : T_j \in F, w_j/p_j < w_h/p_h \}; \]

split \( T_h \) into:

(i) \( T_{h1} \) with \( p_{h1} = t - d^*, \ w_{h1} = p_h w_h/p_h; \)

(ii) \( T_{h2} \) with \( p_{h2} = p_h - p_{h1}, \ w_{h2} = w_h - w_{h1}, \ d_{h2} = d_h; \)

schedule \( T_h \) in \( (d^*, t] \); \( L := L + w_h t; \ CBRK := CBRK + p_h w_h; \)
\[ s := s - p_h; \ t := d^*; \ E := (E \setminus \{ T_h \}) \cup \{ T_{h1} \} \cup \{ T_j \in F : d_j \geq d^* \}; \]
\[ E^- := E \setminus \{ T_j \in F : d_j \geq d^* \}. \]

Correctness of the procedure directly follows from that of the algorithms of Posner [11] and Belouadah et al. [3]. The time complexity is \( O(n \log n) \). Indeed, a splitting can occur only at a deadline, so \( 2n \) pieces at most are scheduled. By using a heap for
set $E$, and observing that set $F$ (introduced for the sake of clarity) needs not be defined explicitly, each iteration requires $O(\log n)$ time.

**Example 1 (continued).** The schedule determined by procedure LB is shown in Fig. 1(b). Step 1 schedules the tasks in $B_{I_1}$ and $B_{I_2}$; we have at this point $s = 24 = \tau$. Step 2 then schedules the remaining tasks in $(0, 24]$ and terminates with $L = -429 + \frac{4}{3}$ and $CBRK = \frac{2}{3}$. At Step 3 we obtain $L = -427$.

### 3. Approximation algorithm

The results of the previous section can be used to obtain an approximation algorithm JOIN, which determines, in polynomial time, a feasible schedule for problem $P$ starting from the optimal solution of problem $SP$ produced by procedure LB. This will also be used to provide an initial upper bound in the exact algorithm presented in the next section. If no task was split (hence $CBRK = 0$) we have an optimal solution to $P$ of value $z^*(P) = L$. Otherwise we can easily obtain a feasible sequence as follows. We start with $t = \max_i\{d_i\}$ and proceed by decreasing completion times until we encounter a piece $T_j$ obtained by splitting a task (or a piece) $T_i$ into $T_{ja}$ and $T_{jb}$, with processing times $p_{ja}$ and $p_{jb}$, respectively, having completion times $C_{ja}$ and $C_{jb}$, with $C_{ja} > C_{jb}$. We can eliminate this infeasibility in three possible ways:

1. by scheduling $T_{ja}$ with completion time $C_{ja} - p_{ja}$ and shifting left by $p_{ja}$ time units all the tasks previously scheduled between $T_{ja}$ and $T_{jb}$;
2. if the total processing time of the tasks preceding $T_{ja}$ is not greater than $C_{ja} - p_{ja} - p_{jb}$, by shifting these tasks left to make the interval $(C_{ja} - p_{ja} - p_{jb}, C_{ja} - p_{ja})$ idle, and scheduling $T_{ja}$ in this interval;
3. if the completion time of each task $T_h$ scheduled between $T_{ja}$ and $T_{jb}$ is not greater than $d_h - p_{ja}$, by shifting right by $p_{ja}$ time units these tasks and scheduling $T_{ja}$ with completion time $C_{ja} + p_{ja}$.

Whenever a piece is encountered, the algorithm evaluates all the above three alternatives and selects the one producing the minimum objective function increase. Since there are at most $n$ splittings, and each iteration requires $O(n)$ time (because of shiftings), the overall time complexity of algorithm JOIN is $O(n^2)$.

The final approximate solution to $P$ is then obtained by optimally inserting idle times through the $O(n)$ procedure described in [2], and post-optimizing through the local search procedure given in [2].

**Example 1 (continued).** The feasible schedule obtained by joining the split tasks is shown in Fig. 1(c). Task $T_9$ is first considered, and alternative (2) is selected; alternative (1) is then selected for $T_8$; the only feasible possibility for $T_6$ is (1). This schedule is then improved through post-optimization, producing the solution of Fig. 1(d). This solution is optimal.
4. Exact solution

We present two exact algorithms for problem P, one based on branch-and-bound and one on dynamic programming.

4.1. Branch-and-bound

Branching strategy

We use a depth-first scheme based on the following branching strategy. The branch-decision tree consists of \(n\) levels, one for each position in the processing sequence, starting from the latest scheduled task and moving backwards. At level \(k\), position \(n - k + 1\) is considered: let \(S\) be the set of tasks currently assigned to positions \(n - k + 2, \ldots, n\); the algorithm generates \(|T\setminus S|\) descending nodes by assigning in turn each unassigned task, according to decreasing deadlines, to the current position and, if possible, by also fixing its completion time. Fixing occurs whenever we can establish that, for the current sequence, the completion time of a task can be optimally determined, according to considerations that will be given later.

At any node, the completion times of the tasks assigned from the root to a certain level have already been optimally fixed, while for those assigned from that level to the current one only the position in the sequence has been defined. Let \(j(k)\) be the index of the task to be assigned at the current level \(k\), and let \(f(k) < k\) be the minimum value, if any, such that the completion time of \(T_{j(k)}\) has not been fixed. We know that: (a) \(w_{j(f(k))} > 0\) since, otherwise, we would have optimally fixed \(C_{j(f(k))} = \min(d_{j(f(k))}, C_{j(f(k) - 1)} - p_{j(f(k) - 1)})\); (b) \(w_{j(f(k) + 1)} + w_{j(f(k))} > 0\), since otherwise \(w_{j(f(k) + 1)} < 0\) so we would have fixed \(C_{j(f(k) + 1)}\) as late as possible by preserving the deadline constraints and then \(C_{j(f(k))}\) as early as possible, i.e., at \(C_{j(f(k) + 1)} + p_{j(f(k))}\); by extending consideration (b) we easily conclude that: (c) \(\sum_{h=f(k)}^{k-1} w_{j(h)} > 0\).

For the current task \(T_{j(k)}\), the algorithm evaluates the minimum and maximum possible completion time: \(t_1(k) = \sum_{j \in T \setminus S} p_j\) and \(t_2(k) = \min(d_{j(k)}, t_2(k - 1) - p_{j(k - 1)})\) (with \(t_2(0) = +\infty\)), respectively. Three cases are then considered:

1. \(t_3(k) < t_1(k)\): the nodes descending from the current one could never produce a feasible solution, so the node is fathomed;
2. \(t_2(k) = t_1(k)\): we fix the completion of \(T_{j(k)}\) at \(t_1(k)\), and then we optimally fix the completion times of all unfixed tasks \(T_{j(h)}\) \((h = k - 1, \ldots, f(k))\) as \(C_{j(h)} = C_{j(h + 1)} + p_{j(h)}\);
3. \(t_3(k) > t_1(k)\): we must decide whether only the position of \(T_{j(k)}\) is defined or whether its completion time too can be optimally fixed. Two possibilities can occur:
   (a) \(w_{j(k)} \geq 0\): \(T_{j(k)}\) should be scheduled as early as possible, so \(C_{j(k)}\) cannot be fixed without information on the tasks which will precede \(T_{j(k)}\) in the sequence,
   (b) \(w_{j(k)} < 0\): we evaluate \(\delta = w_{j(k)} + \sum_{h=f(k)}^{k-1} w_{j(h)}\). If \(\delta > 0\) then \(C_{j(k)}\) cannot be fixed; otherwise we (optimally) schedule \(T_{j(k)}\) as late as possible by preserving the deadline constraints of \(T_{j(h)}\) \((h = k - 1, \ldots, f(k))\), and then we optimally fix the completion times of these tasks too as in case (2).
Fathoming criteria

Before task \(T_{j(k)}\) is assigned, the following fathoming criteria are considered.

**Criterion 1.** If an unassigned task \(T_h\) exists such that \(w_h \leq 0\) and \(t_2(k) + p_h \leq \min(d_{h,k}, t_1(k - 1))\), then the current node can be fathomed. Indeed, any solution having \(T_h\) scheduled earlier than \(T_{j(k)}\) could be improved by moving \(T_h\) to the time interval \((t_2(k), t_2(k) + p_h]\).

**Criterion 2.** If \(d_{j(k)} > t_2(k - 1)\) and \(w_{j(k)}/p_{j(k)} < w_{j(k-1)}/p_{j(k-1)}\), then the current node can be fathomed. The property has been proved in [11, Corollary 2.1].

**Criterion 3.** If an unassigned task \(T_h\) exists such that \(d_h \geq t_2(k), p_h = p_{j(k)}\) and \(w_h < w_{j(k)}\), then the current node can be fathomed. Indeed, any solution having \(T_h\) scheduled earlier than \(T_{j(k)}\) could be improved by interchanging \(T_h\) and \(T_{j(k)}\).

**Criterion 4.** If \(d_h \geq t_2(k) - p_{j(k)}\) for all unassigned tasks \(T_h \neq T_{j(k)}\), then we know that any sequence of these tasks will satisfy the deadline constraints. Hence we can optimally complete the current sequence by assigning \(T_{j(k)}\) to position \(n - k + 1\) and adding the remaining tasks according to nondecreasing values of the ratio \(w_h/p_h\) (Smith's rule [13]), update the incumbent solution and fathom the current node.

Lower bound computation

If the node is not fathomed by the above criteria, a local lower bound value is computed as follows. Let \(F\) be the current set of unassigned tasks (\(T_{j(k)}\) excluded). If the completion time of \(T_{j(k)}\) (hence that of \(T_{j(h)}, h < k\)) has been fixed, the contribution of these tasks to the objective function is known, so the lower bound is simply computed by adding to such value the quantity \(L(F, C_{j(k)} - p_{j(k)})\), where \(L(X, t)\) denotes the value produced by procedure LB when applied to task set \(X\) over time interval \((0, t]\).

If, instead, \(C_{j(k)}\) was not fixed, only contribution \(\tilde{z}\) of the fixed tasks is known; let \(M = \{T_{j(k)}, \ldots, T_{j(f(k))}\}\) be the set of assigned but not fixed tasks, and remember that: (a) \(\sum_{T_j \in M} w_j > 0\); (b) once \(C_{j(k)}\) is fixed the completion times of the remaining tasks of \(M\) can be fixed as \(C_{j(h)} = C_{j(k) + 1} + p_{j(h)} (h = k - 1, \ldots, f(k))\). Hence a valid lower bound on the contribution of task set \(M\) is obtained by assuming \(C_{j(k)} = t_1(k)\) and consequently determining the completion times of the remaining tasks of \(M\); let \(L_M(t_1(k))\) denote such a value. Since no task of \(F\) can be completed after \(t_2(k) - p_{j(k)}\), an overall lower bound for the current node is

\[
L_1 = \tilde{z} + L_M(t_1(k)) + L(F, t_2(k) - p_{j(k)}). \tag{8}
\]

If \(L_1 < UB\), where \(UB\) is the incumbent solution value, the lower bound can be improved as follows. Let \(L(t)\) be the lower bound value we have if \(C_{j(k)} = t\) (with \(t_1(k) \leq t \leq t_2(k)\)), i.e., \(L(t) = \tilde{z} + L_M(t) + L(F, t_2(k) - p_{j(k)})\), observe that \(L(t)\) is
nondecreasing in $t$, and let

$$t = \max \{ t: L(t) < UB \}. \quad (9)$$

If $t < t_2(k)$, we know that the tasks in $F$ can only be scheduled in $(0, t - p_{j(k)})$, so

$$L_2 = \bar{t} + L_M(t_1(k)) + L(F, t - p_{j(k)}) \quad (10)$$

is a better lower bound value. The process can be iterated until either $L_i \geq UB$, or $L_i = L_{i-1}$ (indeed, from (8)–(10), the sequence of $L_i$ values is nondecreasing).

4.2. Dynamic programming

We have used a standard dynamic programming recursion for sequencing problems (see, e.g., Bard et al. [2], Held and Karp [8]). For any $X \subseteq T$ and time instant $t$, the state $(X, t)$ denotes the optimal solution to the subproblem of $P$ defined by task set $X$ with the additional constraint that the maximum completion time is exactly $t$. Let $f(X, t)$ be the value of state $(X, t)$, with $f(X, t) = +\infty$ if the subproblem has no feasible solution. We trivially have $f(\emptyset, 0) = 0$. The algorithm considers $n$ stages (for $k = |X|$ increasing from 1 to $n$) and computes at each stage the values $f(X, t)$ through the recursion:

$$f(X, t) = \min_{T_j \in X: d_j \geq t} \left\{ \min_{9 \leq t - \sum_{h \in Y \setminus \{T_j\}}} \left\{ f(X \setminus \{T_h\}, 9) + w_{ht} \right\} \right\}, \quad X \subseteq T, |X| = k, t \leq \max_j \{d_j\}.$$

The number of states $(X, t)$ is bounded by $2^n \max_j \{d_j\}$; for each state, the computation of $f(X, t)$ requires $O(n \max_j \{d_j\})$ time. The overall time complexity is thus $O(n^2 \max_j \{d_j\})^2$.

At stage $k$, for each $T_j \in T$ and each set $Y \in \{X \subseteq T: |X| = k - 1\}$ and $X \not\supseteq T_j$, let us consider the time instants $9 = \sum_{T_h \in Y} p_h, \ldots, \max \{d_h: T_h \in Y\},$ then for $t = 9 + p_j, \ldots, t_{\text{max}}$ (where $t_{\text{max}} \leq d_j$ will be defined later)

$$z((Y, 9), T_j, t) = f(Y, 9) + w_{jt} \quad (11)$$

is the value of the optimal solution (subschedule) to the subproblem of $P$ consisting of task set $Y \cup \{T_j\}$ with task $T_j$ scheduled last at $C_j = t$. Hence for each $X$ such that $|X| = k$, we have $f(X, t) = \min\{z((Y, 9), T_j, t): Y \cup \{T_j\} = X\}$.

The value $t_{\text{max}}$ can be determined by scheduling, according to the EDD rule (see Section 1), the set of “free” tasks $F = T \setminus (Y \cup \{T_j\})$ in time interval $(9 + p_j, +\infty)$ and finding in $F$ the minimum difference $\delta$ between a deadline and the completion time of the corresponding task. Hence $t_{\text{max}} = 9 + p_j + \delta$, since we cannot schedule all tasks of $F$ in time interval $(t_{\text{max}} + 1, +\infty)$ if $C_j > t_{\text{max}}$.

The number of subschedules to be generated at each stage can be reduced through dominance criteria. Let $T_i \in Y$ be the task with $C_i = 9$ in state $(Y, 9)$, and $T_j$ the task currently considered for extending the state. The following two criteria have been introduced in [2].
**Criterion 5.** If \( w_j \geq 0 \), for any \((Y, \theta)\) only \( z((Y, \theta), T_j, \theta + p_j) \) needs to be computed. If, in addition, \( w_i < 0 \) and \( d_i \geq \theta + p_j \), no subschedule at all is generated for \( T_j \).

**Criterion 6.** If \( w_j < 0 \), for any \((Y, \theta)\) such that \( w_i < 0 \) and \( d_i > \theta + p_j \) only \( z((Y, \theta), T_j, \theta + p_j) \) needs to be computed.

Given a feasible solution with value \( UB \), the number of subschedules can be further reduced through local lower bound computations. Let \( L'(F, t) \) denote the lower bound value produced by procedure LB when applied to task set \( F \) over time interval \((t, + \infty)\). The following new criteria have computationally proved to be highly effective.

**Criterion 7.** No subschedule needs to be generated for any pair \((T_j, (Y, \theta))\) such that \( w_i/p_i < w_j/p_j \) and \( d_i \geq \theta + p_j \). Indeed, from Criteria 5 and 6 we know that the only nondominated subschedule would have \( t = \theta + p_j \), but any solution obtained from it can be improved by interchanging \( T_i \) and \( T_j \).

**Criterion 8.** If \( L'(F, t) \) is produced by procedure LB with no task splitting, the subschedule is not generated, since we have its optimal completion (hence the incumbent solution is possibly updated).

**Criterion 9.** If, for the current \( t \), \( z((Y, \theta), T_j, t) + L'(F, t) \geq UB \) holds, then the subschedule needs not to be generated. If in addition the current \( T_j \) has \( w_j \geq 0 \), then the computation of (11) can be halted since both terms of the left-hand side are non-decreasing as \( t \) increases, so no higher value of \( t \) could produce nondominated subschedules.

**Criterion 10.** If \( w_h \leq 0 \) for each \( T_h \in F \), then \( L'(F, t) \) takes the same value for all \( t = \theta + p_j, \ldots, t_{\text{max}} \) hence needs to be computed only once for each \( T_j \) and \((Y, \theta)\). In addition, if \( w_j < 0 \) it is convenient to execute the computation of (11) for \( t = t_{\text{max}}, t_{\text{max}} - 1, \ldots \), terminating as soon as the local lower bound is no less than \( UB \) (while if \( w_j \geq 0 \), the computation is terminated by Criterion 9).

**Criterion 11.** If a task \( T_h \in F \) exists such that \( p_h \geq p_j, d_h \leq d_j \), and \( w_h \geq w_j \), then the maximum value of \( t \) to be considered for the computation of (11) is \( \theta + p_h - 1 \). Indeed, any solution with \( C_h > C_j \) having an idle time interval of length at least \( p_h - p_j \) just before \( T_j \) could be improved by interchanging \( T_j \) and \( T_h \).

**Criterion 12.** For any \((Y, \theta)\) and \( T_j \), no \( z((Y, \theta), T_j, t) \) such that \( t - \theta - p_j \geq p_h^* = \min\{p_h : T_h \in F, w_h \geq 0\} \) needs to be computed. Indeed, any solution obtained from such a subschedule (i.e., having \( C_j \gg \theta + p_j + p_h^* \) and \( C_h^* > C_j \)) could be improved by moving \( T_h^* \) to the idle time interval \((\theta, \theta + p_h^*)\).
We finally observe that precedence relations can exist between tasks with equal processing time. If $T_s$ and $T_r$ are such that $p_s = p_r$, $d_s \leq d_r$, and $w_s \geq w_r$, then any solution to $P$ with $C_s < C_r$ could be improved by interchanging $T_s$ and $T_r$. Hence $P_j = \{ T_h \neq T_j: p_h = p_j, d_h \leq d_j, \text{and } w_h \geq w_j \}$ is the set of tasks which needs to be scheduled before $T_j$ in an optimal solution (when all relations hold with equality, $T_h \in P_j$ only if $h < j$). It follows that, at any stage, no subproblem is generated for those $T_j$ and $Y$ such that $P_j \notin Y$.

5. Computational experiments

We coded in C language the branch-and-bound algorithm BB of Section 4.1 and the dynamic programming algorithm DP of Section 4.2. We executed computational experiments on a PC 486 with a 33 MHz clock by using the data generations described, for problem $P'$, in [2, 5]. For each task $T_j$ the values of $p_j$, $a_j$, and $b_j$ are uniformly randomly generated in range $[1, \ell_0]$ so that

- **Class 1:** $a_j \leq b_j$ for approximately 50% of the tasks;
- **Class 2:** $a_j \leq b_j$ for approximately 66% of the tasks;
- **Class 3:** $a_j \leq b_j$ for approximately 33% of the tasks.

The deadline of each task $T_j$ is uniformly randomly generated in range $[\beta^- \sum_{i=1}^{j-1} p_i, \beta^+ \sum_{i=1}^{j-1} p_i]$, with four $(\beta^-, \beta^+)$ pairs: $(0.75, 1.25)$, $(0.25, 1.75)$, $(0.75, 1.75)$ and $(0.50, 2.50)$.

For each pair $(\beta^-, \beta^+)$ in each class and for each value of $n (= 10, 20, 30, 40, 50)$, five feasible problems were generated, giving a total of 300 instances.

The results are given in columns BB and DP of Tables 1–3. The branch-and-bound algorithm was executed with an imposed limit $M$ on the number of decision nodes (experimentally determined as $M = 5n10^3$), the dynamic programming algorithm with a limit of 16000 on the number of states. The entries give the average CPU times and, in brackets, the number of unsolved instances, if any. The results show a certain "complementarity" of the two approaches in the sense that several instances were very difficult for one of them but relatively easy for the other. Such behavior is not unusual when NP-hard problems of limited size are solved through depth-first branch-and-bound or dynamic programming algorithms. Indeed, if an instance has many equivalent solutions the branch-and-bound approach tends to explore a high number of decision nodes, while dynamic programming easily eliminates equivalent states. If, instead, the instance has few equivalent solutions and the branch-and-bound algorithm finds an optimum after a few decision nodes, a tight bound can fathom most of the remaining nodes, while dynamic programming needs in any case to perform all the states before a complete solution is obtained. Similar behavior is encountered, for example, in the solution of knapsack-type problems (see, e.g. Martello and Toth [10]).

Hence we obtained an effective exact algorithm BBDP by combining the two approaches as follows. We start by executing procedures LB and JOIN; if the lower and upper bound are different, the branch-and-bound algorithm is executed: if the
Table 1
Class 1: $a_j \leq \beta_j$ for approximately 50% of the tasks

<table>
<thead>
<tr>
<th>$(\beta^-, \beta^+)$</th>
<th>$n$</th>
<th>Exact solution</th>
<th>Approximate solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BB</td>
<td>DP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time</td>
</tr>
<tr>
<td>$(0.75, 1.25)$</td>
<td>10</td>
<td>0.02</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.05</td>
<td>0.41</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1.65</td>
<td>13.20</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>48.83</td>
<td>25.83</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>147.88</td>
<td>135.63 (3)</td>
</tr>
<tr>
<td>$(0.25, 1.75)$</td>
<td>10</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.06</td>
<td>0.76</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>6.42</td>
<td>19.33</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>23.87</td>
<td>33.14 (1)</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>178.93 (2)</td>
<td>106.04 (1)</td>
</tr>
<tr>
<td>$(0.75, 1.75)$</td>
<td>10</td>
<td>0.01</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.24</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>4.23</td>
<td>3.52</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>70.60</td>
<td>28.16 (1)</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>160.61 (1)</td>
<td>109.23 (2)</td>
</tr>
<tr>
<td>$(0.50, 2.50)$</td>
<td>10</td>
<td>0.02</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.18</td>
<td>0.29</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>5.54</td>
<td>1.63</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>123.86 (1)</td>
<td>20.68</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>170.54 (1)</td>
<td>56.85</td>
</tr>
</tbody>
</table>

Table 2
Class 2: $a_j \leq \beta_j$ for approximately 66% of the tasks

<table>
<thead>
<tr>
<th>$(\beta^-, \beta^+)$</th>
<th>$n$</th>
<th>Exact solution</th>
<th>Approximate solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BB</td>
<td>DP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time</td>
</tr>
<tr>
<td>$(0.75, 1.25)$</td>
<td>10</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.26</td>
<td>1.68</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>3.27</td>
<td>4.60</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>103.88 (1)</td>
<td>27.24</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>301.06 (3)</td>
<td>107.26 (4)</td>
</tr>
<tr>
<td>$(0.25, 1.75)$</td>
<td>10</td>
<td>0.00</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.27</td>
<td>0.80</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1.77</td>
<td>7.24</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>126.88 (1)</td>
<td>20.53 (1)</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>185.05 (1)</td>
<td>120.69 (2)</td>
</tr>
</tbody>
</table>
Table 2 continued

<table>
<thead>
<tr>
<th>$(\beta^-, \beta^+)$</th>
<th>$n$</th>
<th>Exact solution</th>
<th>Approximate solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BB</td>
<td>DP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time</td>
</tr>
<tr>
<td>10</td>
<td>0.03</td>
<td>0.04</td>
<td>0.03</td>
</tr>
<tr>
<td>20</td>
<td>0.48</td>
<td>0.53</td>
<td>0.48</td>
</tr>
<tr>
<td>$(0.75, 1.75)$</td>
<td>30</td>
<td>2.26</td>
<td>23.27 (1)</td>
</tr>
<tr>
<td>40</td>
<td>16.96</td>
<td>27.26 (1)</td>
<td>16.96</td>
</tr>
<tr>
<td>50</td>
<td>159.19 (1)</td>
<td>41.23 (1)</td>
<td>197.79</td>
</tr>
<tr>
<td>10</td>
<td>0.04</td>
<td>0.05</td>
<td>0.04</td>
</tr>
<tr>
<td>20</td>
<td>0.34</td>
<td>0.64</td>
<td>0.34</td>
</tr>
<tr>
<td>$(0.50, 2.50)$</td>
<td>30</td>
<td>14.31</td>
<td>5.86</td>
</tr>
<tr>
<td>40</td>
<td>102.93 (1)</td>
<td>31.76</td>
<td>103.73</td>
</tr>
<tr>
<td>50</td>
<td>278.05 (3)</td>
<td>45.46</td>
<td>306.65</td>
</tr>
</tbody>
</table>

Table 3

Class 3: $\alpha_j \leq \beta_j$ for approximately 33% of the tasks

<table>
<thead>
<tr>
<th>$(\beta^-, \beta^+)$</th>
<th>$n$</th>
<th>Exact solution</th>
<th>Approximate solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BB</td>
<td>DP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time</td>
</tr>
<tr>
<td>10</td>
<td>0.02</td>
<td>0.00</td>
<td>0.02</td>
</tr>
<tr>
<td>20</td>
<td>0.13</td>
<td>0.41</td>
<td>0.13</td>
</tr>
<tr>
<td>$(0.75, 1.25)$</td>
<td>30</td>
<td>2.12</td>
<td>4.35</td>
</tr>
<tr>
<td>40</td>
<td>21.93</td>
<td>67.00 (2)</td>
<td>21.93</td>
</tr>
<tr>
<td>50</td>
<td>217.82 (2)</td>
<td>116.00 (4)</td>
<td>343.82 (2)</td>
</tr>
<tr>
<td>10</td>
<td>0.04</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>20</td>
<td>0.26</td>
<td>0.64</td>
<td>0.26</td>
</tr>
<tr>
<td>$(0.25, 1.75)$</td>
<td>30</td>
<td>6.59</td>
<td>3.47</td>
</tr>
<tr>
<td>40</td>
<td>29.01</td>
<td>39.14</td>
<td>29.01</td>
</tr>
<tr>
<td>50</td>
<td>230.91 (2)</td>
<td>144.18 (3)</td>
<td>300.51 (1)</td>
</tr>
<tr>
<td>10</td>
<td>0.04</td>
<td>0.02</td>
<td>0.04</td>
</tr>
<tr>
<td>20</td>
<td>0.17</td>
<td>0.44</td>
<td>0.17</td>
</tr>
<tr>
<td>$(0.75, 1.75)$</td>
<td>30</td>
<td>2.97</td>
<td>4.29</td>
</tr>
<tr>
<td>40</td>
<td>16.55</td>
<td>79.79 (3)</td>
<td>16.55</td>
</tr>
<tr>
<td>50</td>
<td>253.23 (3)</td>
<td>87.87 (2)</td>
<td>316.03</td>
</tr>
<tr>
<td>10</td>
<td>0.05</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>20</td>
<td>0.60</td>
<td>0.69</td>
<td>0.60</td>
</tr>
<tr>
<td>$(0.50, 2.50)$</td>
<td>30</td>
<td>3.68</td>
<td>3.09</td>
</tr>
<tr>
<td>40</td>
<td>221.37 (2)</td>
<td>39.41</td>
<td>242.37</td>
</tr>
<tr>
<td>50</td>
<td>251.71 (3)</td>
<td>60.01</td>
<td>290.71</td>
</tr>
</tbody>
</table>
optimal solution is not found within $M$ decision nodes, the dynamic programming algorithm is executed (with a limit of 16,000 states), using as upper bound the best solution obtained so far. Columns BBDP of Tables 1–3 show that the resulting algorithm effectively solved the generated instances. The dynamic programming phase was executed in 28 cases out of 300. The performance of BBDP compares favorably with that of the dynamic programming approach tested in [2].

We also coded in C language the approximation algorithm JOIN of Section 3 and the GRASP heuristic described in [5]. The corresponding computational results are in the last two columns of Tables 1–3, where each entry gives the average CPU time and the average percentage error with respect to the optimal solution. Algorithm JOIN produced very good solutions with CPU times one order of magnitude smaller than the GRASP approach.

6. Conclusion

The scheduling problem considered in this paper generalizes a classical problem in scheduling theory and has interesting applications in Just In Time production. It is NP-hard and in practice very difficult to solve. We have developed a bounding scheme and a hybrid algorithm which permits the exact solution of instances having up to 50 tasks, the largest size ever attained. We have also obtained an approximation algorithm which outperforms heuristic algorithms from the literature.

Acknowledgements

This research was supported by the Ministero dell'Università e della Ricerca Scientifica e Tecnologica and by the Consiglio Nazionale delle Ricerche, Italy. Thanks are due to two anonymous referees for their valuable comments.

References


