Single Machine Scheduling Subject to Precedence Delays

Lucian FINTA   Zhen LIU

INRIA, Centre Sophia Antipolis
2004 route des Lucioles
B.P. 93
06902 Sophia-Antipolis, France

December 1993, Revised January 1995 and March 1995

Abstract

A single-machine scheduling problem with precedence delays is analyzed. A set of n tasks is to be scheduled on the machine in such a way that the makespan is minimized. The executions of the tasks are constrained by precedence delays, i.e., a task can start its execution only after any of its predecessors has completed and the delay between the two tasks has elapsed. In the case of unit execution times and integer lengths of delays, the problem is shown to be NP-hard in the strong sense. In the case of integer execution times and unit length of delays, the problem is polynomial, and an $O(n^2)$ optimal algorithm is provided. Both preemptive and non-preemptive cases are considered.

Keywords: Scheduling, Makespan, Precedence Delay, Release Time, Delivery Time, Complexity, Optimal Algorithm.

1 Introduction and Problem Description

Consider the following scheduling problem. There are a single machine and a set of n tasks to be run on that machine. The executions of the tasks are constrained by precedence constraints which are described by a directed acyclic graph $G = (V, E)$, referred to as task graph, where the set of vertices V corresponds to the set of tasks and the set of arcs E to the precedence constraints. The task graph is a weighted graph with vertices weighted by task processing times p_i, $i \in V$, and arcs weighted by lengths of delays l_{ij}. For any pair of tasks $i, j \in V$, if $(i, j) \in E$, then task j can start execution only l_{ij} time units after the execution completion of task i, i.e., $C_i + l_{ij} \leq \sigma(j)$, where C_i is the completion time of task i, $\sigma(j)$ is the starting time of task j. Throughout the paper, processing times and precedence delays are assumed to be nonnegative integers. The problem is to find a feasible schedule (which satisfies the precedence delays) such that the makespan $C_{\max}$, i.e. the completion time of the last executed task, $C_{\max} = \max_{i \in V} C_i$, is minimized. We analyze both the preemptive and the non-preemptive scheduling problems.

According to the three-field notation scheme introduced by Graham, Lawler, Lenstra and Rinnooy Kan [12], our non-preemptive (resp. preemptive) scheduling problem can be denoted as $1 \mid prec(l_{ij}),\ p_j \mid C_{\max}$ (resp. $1 \mid pmtn,\ prec(l_{ij}),\ p_j \mid C_{\max}$), where l_{ij} denotes precedence delays. In case we have identical parallel machines, the problems can be denoted by $P \mid prec(l_{ij}),\ p_j \mid C_{\max}$ and $P \mid pmtn,\ prec(l_{ij}),\ p_j \mid C_{\max}$.

The notion of precedence delays can be used to model the release date of the tasks. Indeed, by adding a fictive "initial" task of which all tasks are its successors, the precedence delay between the "initial" task and any particular task can be considered as the release date of the latter. In a similar way, the precedence delays can also be used to model the delivery times (which are in certain sense equivalent to due dates). Indeed, by adding a fictive "final" task of which all tasks are its predecessors, the precedence delay between a task and the "final" task can be considered as the delivery time of the former.

Note however that the notion of precedence delays is different from that of communication times in the scheduling literature (see e.g. [18]). The effective communication times between tasks depend on the task assignment. Communication times between tasks which are assigned to the same machine are usually assumed to be small, typically negligible (i.e. zero). However, precedence delays between tasks are assumed to remain unchanged even when two tasks are assigned to the same machine.

Single-machine scheduling has been receiving much interest in the literature. Indeed, as Baker [2] indicated, it is a building block in the development of a comprehensive understanding of complicated systems. The reader is referred to the survey papers by Dileepan and Sen [8], Gupta and Kyparisis [13], and Lawler, Lenstra, Rinnooy Kan and Shmoys [15] for research work in this field.

The scheduling problem analyzed in this paper is an extension of the model with release and delivery times (or due dates). Moreover, it has direct applications in manufacturing systems and computer systems. For example, one of the first studies of scheduling under precedence delays is due to Dauzère-Peres and Lasserre [6, 7], where the authors used a solution of the single-machine scheduling to solve the job-shop scheduling problem. They proposed a modification of the shifting bottleneck procedure of Adams et al. [1]. Such a modification takes into account the precedence delays associated with the precedence relations induced by scheduling a bottleneck machine, and therefore yields better performances. In [9], a scheduling problem of multiprocessor system is reduced to this single-machine model, where the tasks represent communications on a bus and the precedence delays represent execution times of threads in parallel processors.

The general non-preemptive scheduling problem for makespan minimization subject to release and delivery times (which corresponds to the problem with precedence delays of zero length in-between tasks except for those associated with the "initial" and the "final" tasks) was shown to be NP-hard by Garey and Johnson [10]. Carlier [4] proposed an efficient branch-and-bound algorithm for solving the problem. When all the task processing times are equal, Simons [17] and Garey, Johnson, Simons and Tarjan [11] proposed polynomial algorithms for the optimal solution.

In the preemptive case, however, simple polynomial algorithms solve the problem for makespan minimization subject to release and delivery times. Indeed, as observed by Garey et al. [11], the presence of precedence constraints (with zero delay) is essentially irrelevant in this case: One can first modify the release and delivery times so that they become consistent with the precedence relations, and then apply the Largest-Delivery-Time policy, see Horn [14].

It is easily seen from the above discussions that the NP-hardness of makespan minimization subject to release and delivery times implies the NP-hardness of makespan minimization subject to integer precedence delays. Balas, Lenstra and Vazacopoulos [3] (where the term "delayed precedence constraints" was used) showed that when release and delivery times are all equal, the makespan minimization subject to integer lengths of precedence delays remains NP-hard even if preemption is allowed.

In this paper, we show that even if tasks have unit execution time (UET), the problem of makespan minimization subject to integer lengths of precedence delays is still NP-hard (in the strong sense). However, in case of unit length of precedence delay (UPD), even if the tasks have arbitrary integer execution times, the problem becomes polynomial, and we provide an $O(n^2)$ algorithm. These results hold for both preemptive and non-preemptive scheduling.

The presentation of the paper is organized as follows. In Sections 2 and 3 below, we consider non-preemptive scheduling problems. We prove in Section 2 the NP-hardness for the case of arbitrary integer precedence delays. In Section 3, we provide the polynomial solution for the case of unit precedence delay. In Section 4, we extend these results to preemptive scheduling problems. Finally, in Section 5, we provide some concluding remarks.

2 NP-hardness

In this section, we prove the NP-hardness of the non-preemptive scheduling problem $1 \mid prec(l_{ij}),\ p_j = 1 \mid C_{\max}$. We consider the associated decision problem defined as follows.

(P1): Single-machine scheduling with unit execution time and integer lengths of precedence delays. Given a directed acyclic graph $G = (V, E)$ with unit execution time $p_j = 1$ for all $j \in V$, precedence delays $l_{ij} \in I\!N_+ \stackrel{\text{def}}{=} \{1, 2, \cdots\}$, and a time limit $T \in I\!N_+$, does there exist a function $\sigma : V \to \{0, 1, \cdots, T-1\}$ such that $\sigma(i) + 1 + l_{ij} \leq \sigma(j)$ for all $(i, j) \in E$?

This problem will be shown to be NP-complete. In order to do that, we begin by introducing a slightly more complex problem (P2) which can be polynomially transformed to (P1). In (P2) there are some forbidden regions for the scheduling function, i.e., the machine is not available in some periods of time. We then show this new problem (P2) to be NP-complete so that (P1) is also NP-complete.

(P2): Single-machine scheduling with unit execution time, integer lengths of precedence

delays and forbidden regions. Given a directed acyclic graph $G = (V, E)$ with unit execution time $p_j = 1$ for all $j \in V$, precedence delays $l_{ij} \in I\!N_+$, a time limit $T \in I\!N_+$, $T \leq 2|V|$, and some positive integers $0 \leq b_1 < e_1 < b_2 < e_2 < \cdots < b_r < e_r < T$, does there exist a function $\sigma : V \rightarrow \{0, 1, \cdots, T-1\}$ such that

(i) for all $(i, j) \in E$, $\sigma(i) + 1 + l_{ij} \leq \sigma(j)$, and

(ii) for all $i \in V$ and all $s \in \{1, 2, \cdots, r\}$, $\sigma(i) \notin [b_s, e_s)$?

Note that the time limit in (P2) is polynomial in the number of vertices of the graph to be scheduled, i.e. $T \leq 2|V|$.

Lemma 1 *(P2) polynomially transforms to (P1).*

Proof. Let

$$V^\circ = \bigcup_{1 \leq s \leq r} \{b_s, b_s + 1, \cdots, e_s - 2, e_s - 1\}.$$

Let $V' = \{i_1, i_2, \cdots, i_h\}$ (resp. $V'' = \{j_1, j_2, \cdots, j_k\}$) be the odd (resp. even) numbers in V° such that $V^\circ = V' \cup V''$ and $i_1 < i_2 < \cdots < i_h$, $j_1 < j_2 < \cdots < j_k$. We construct two chains $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ such that

$$\begin{aligned} V_1 &= \{v_{i_1}, v_{i_2}, \cdots, v_{i_h}\}, \\ E_1 &= \{(v_{i_s}, v_{i_{s+1}}) \mid s = 1, 2, \cdots, h-1\}, \\ V_2 &= \{v_{j_1}, v_{j_2}, \cdots, v_{j_k}\}, \\ E_2 &= \{(v_{j_s}, v_{j_{s+1}}) \mid s = 1, 2, \cdots, k-1\}. \end{aligned}$$

The lengths of precedence delays are defined as follows:

$$\begin{aligned} l_{v_{i_s}, v_{i_{s+1}}} &= i_{s+1} - i_s - 1, \qquad s = 1, 2, \cdots, h-1, \\ l_{v_{j_s}, v_{j_{s+1}}} &= j_{s+1} - j_s - 1, \qquad s = 1, 2, \cdots, k-1. \end{aligned}$$

Now, for any given instance of (P2) with task graph G and time limit T, we construct an instance of (P1) as follows. The task graph G' in (P1) is defined as the union of G, G_1 and G_2 connected by a initial task a_1 and a final task a_2 in such a way that

- the precedence delays between a_1 and any task of G without predecessors are 1;
- the precedence delays between any tasks of G without successors and a_2 are 1;
- the precedence delay between a_1 and v_{i_1} (resp. v_{j_1}) is $i_1 + 1$ (resp. $j_1 + 1$);
- the precedence delay between v_{i_h} (resp. v_{j_k}) and a_2 is $T - i_h + 1$ (resp. $T - j_k + 1$).

The time limit in (P1) is T+4.

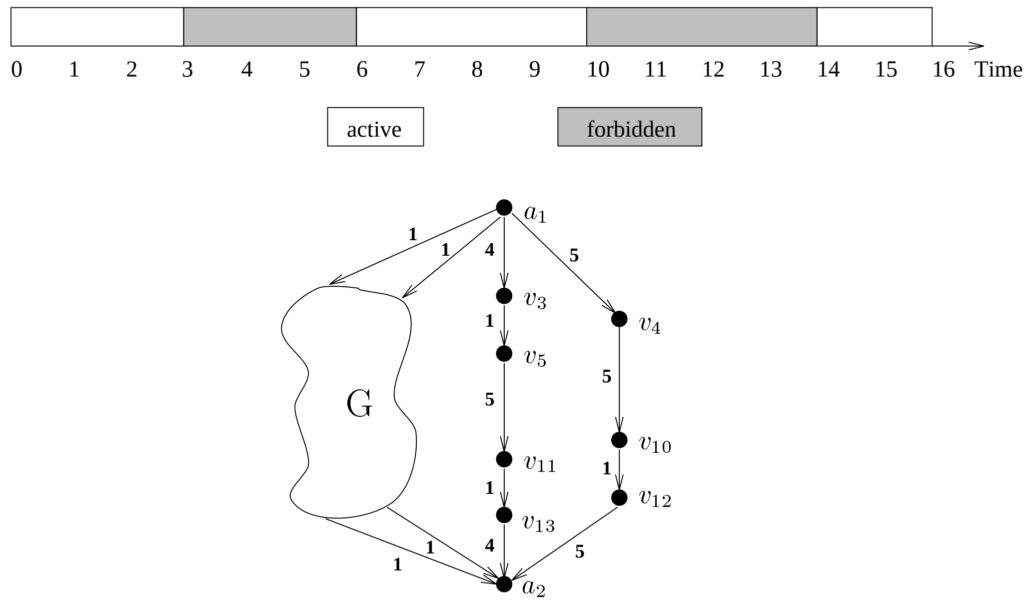

Figure 1: Construction of G' given G and the forbidden regions.

As an example, consider the the forbidden regions in Figure 1. There are two forbidden regions with $b_1 = 3$, $e_1 = 6$, $b_2 = 10$, $e_2 = 14$, and the time limit is $T = 16$ for (P2). Therefore, the set V° contains the numbers $3, 4, 5, 10, 11, 12, 13$. It follows that $V' = \{3, 5, 11, 13\}$ and $V'' = \{4, 10, 12\}$. Thus, $V_1 = \{v_3, v_5, v_{11}, v_{13}\}$ and $V_2 = \{v_4, v_{10}, v_{12}\}$. The chain G_1 (resp. G_2) is simply $v_3 \to v_5 \to v_{11} \to v_{13}$ (resp. $v_4 \to v_{10} \to v_{12}$), where the symbol "$\to$" represents an arc. The time limit for the constructed instance of problem (P1) is 20 (for the schedule of G').

It is easy to see that there is a solution to (P2) if and only if there is a solution to (P1). Indeed, according to the construction of G', the chains G_1 and G_2 are critical paths in G' so that task v_{i_s}, $1 \le s \le h$, (resp. v_{j_s}, $1 \le s \le k$) should be executed at time $i_s + 2$ (resp. $j_s + 2$). ∎

We now transform the classical 3-satisfiability problem, denoted by 3SAT, to (P2) by a polynomial transformation. Recall the definition of 3SAT:

3SAT: 3-*satisfiability*. Given a set X of binary variables x_i, $1 \le i \le m$, and a collection C of clauses C_j over X, $1 \le j \le k, |C_j| = 3$, is there a satisfying truth assignment for C?

Lemma 2 *3SAT polynomially transforms to (P2).*

Proof. Given an instance of 3SAT as above, we construct the following instance of (P2), such that there exists a scheduling function σ if and only if 3SAT has a solution.

The structure of our task graph $G = (V, E)$ is similar to the one used by Ullman [16] in the proof of NP-hardness of makespan minimization of UET tasks on identical machines under precedence constraints. However, since we have only one machine, the one-to-one correspondence between the solutions of these problems is much more difficult to prove.

Let us start with an informal definition of the task graph G to be constructed. For each variable $x_i \in X$, $1 \le i \le m$, we have two paths $x_{i,0} \to x_{i,1} \to \cdots \to x_{i,m}$ and $\bar{x}_{i,0} \to \bar{x}_{i,1} \to \cdots \to \bar{x}_{i,m}$ in G. The arcs $(x_{i,j-1}, x_{i,j})$ and $(\bar{x}_{i,j-1}, \bar{x}_{i,j})$ have precedence delays $l_{x_{i,j-1},x_{ij}} = l_{\bar{x}_{i,j-1},\bar{x}_{ij}} = 2m + j$ for $1 \le i \le m$, $1 \le j \le m$. For each path there is one more vertex y_i or $\bar{y}_i$, without outgoing arc, connected to the path by an arc $(x_{i,i-1}, y_i)$ or $(\bar{x}_{i,i-1}, \bar{y}_i)$ with precedence delays $l_{x_{i,i-1},y_i} = l_{\bar{x}_{i,i-1},\bar{y}_i} = m$ for $1 \le i \le m$. For each clause $C_r \in C$, $1 \le r \le k$, we have in G seven vertices c_{rs}, $1 \le s \le 7$. There is an arc from $x_{i,m}$ (or $\bar{x}_{i,m}$) to each clause vertex $c_{r,s}$ whenever it contributes.

An example of the construction of the instance of (P2) is illustrated in Figure 2, where the set of literals is $X = \{x_1, x_2, x_3, x_4\}$ and the clauses are $C_1 = x_1 + x_2 + \bar{x}_3$ and $C_2 = \bar{x}_1 + x_3 + \bar{x}_4$, hence, $k = 2$, $m = 4$. The graph is top-bottom oriented and all the arcs at the same level have the same length. Some vertices and vertex names, and some arcs connected to clause vertices $c_{2,j}$ are omitted for sake of simplicity, $1 \le j \le 7$.

The formal definition of the graph is the following.

- The set of vertices V contains:
  - x_{ij} and $\bar{x}_{ij}$ for $1 \le i \le m$, $0 \le j \le m$,
  - y_i and $\bar{y}_i$ for $1 \le i \le m$,

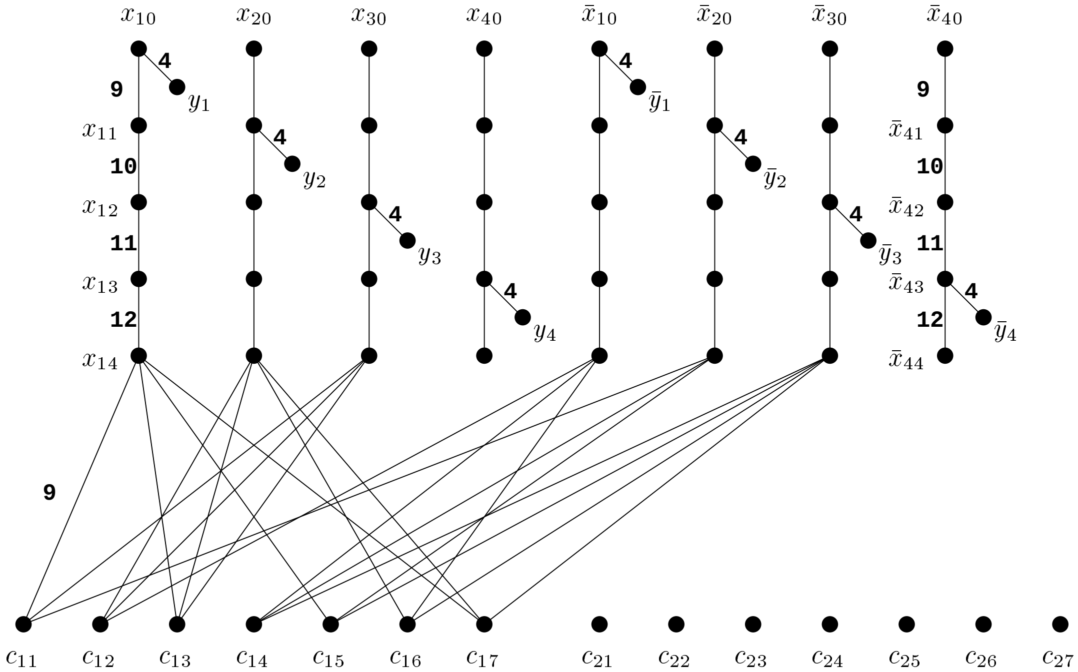

Figure 2: Task graph of the instance (P2) corresponding to the instance of 3SAT.

  - c_{rs} for $1 \leq r \leq k$ and $1 \leq s \leq 7$.

- The set of arcs E and precedence delays are:

  - $(x_{i,j-1}, x_{ij})$ and $(\bar{x}_{i,j-1}, \bar{x}_{ij})$ with precedence delays $l_{x_{i,j-1},x_{ij}} = l_{\bar{x}_{i,j-1},\bar{x}_{ij}} = 2m + j$ for $1 \leq i \leq m$, $1 \leq j \leq m$,
  - $(x_{i,i-1}, y_i)$ and $(\bar{x}_{i,i-1}, \bar{y}_i)$ with precedence delays $l_{x_{i,i-1},y_i} = l_{\bar{x}_{i,i-1},\bar{y}_i} = m$ for $1 \leq i \leq m$.
  - The arcs connecting x_{im} (resp. $\bar{x}_{im}$) and c_{rs}, $1 \leq i \leq m$, $1 \leq r \leq k$, $1 \leq s \leq 7$, are defined as follows. Let C_r consist of literals $z_{u_1}, z_{u_2}, z_{u_3}$, where each z independently stands for x or $\bar{x}$, in a fixed order, i.e. $C_r = z_{u_1} + z_{u_2} + z_{u_3}$, $1 \leq u_1 < u_2 < u_3 \leq m$, $1 \leq r \leq k$. Let $a_3a_2a_1$ be the binary representation of s, $1 \leq s \leq 7$. Then for $1 \leq p \leq 3$, if $a_p = 1$, we have an arc $(z_{u_p m}, c_{rs})$, else, if $a_p = 0$, then we have an arc $(\bar{z}_{u_p m}, c_{rs})$, where $\bar{z}$ stands for $\bar{x}$ or x, should z be x or $\bar{x}$, respectively. Note that since a clause should have at least one true literal, the case $a_1 = a_2 = a_3 = 0$ cannot occur.
  - The precedence delays are defined by $l_{z_{u_p m},c_{rs}} = l_{\bar{z}_{u_p m},c_{rs}} = k + 2m - 1$.

There are $m + 2$ forbidden regions for the scheduling function σ. For $1 \leq i \leq m - 1$, the i-th forbidden region $F_i = [b_i, e_i)$ is of length i. The last three forbidden regions $F_i = [b_i, e_i)$, $m \le i \le m+2$, are of lengths k, $m-1$, m, respectively. The start and the end of those regions are:

$$\begin{array}{rclrcl}
b_i &=& 2m+1+(2m+2)i+\frac{i(i-1)}{2}, & e_i &=& 2m+1+(2m+2)i+\frac{i(i+1)}{2}, \quad 1 \le i \le m-1,\\
b_m &=& \frac{5m^2+5m}{2}, & e_m &=& \frac{5m^2+5m}{2}+k,\\
b_{m+1} &=& \frac{5m^2+7m}{2}+k, & e_{m+1} &=& \frac{5m^2+9m}{2}+k-1,\\
b_{m+2} &=& \frac{5m^2+9m}{2}+2k-1, & e_{m+2} &=& \frac{5m^2+11m}{2}+2k-1.
\end{array}$$

Thus, there are $m+3$ active regions for the machine, the first one with length $4m+3$, the next $m-2$ ones with length $2m+2$, the m-th with length $m+1$, the $m+1$-st with length m, the $m+2$-nd with length k and the $m+3$-rd with length $6k$.

The time limit is $T = \frac{5m^2+11m}{2} + 8k - 1$. Observe that $|V| = 2m(m+1) + 2m + 7k$. Thus,

$$\begin{aligned}
T &= \frac{5m^2+11m}{2} + 8k - 1\\
&\le 3m(m+1) + 3m + 8k - 1\\
&\le 2|V|
\end{aligned}$$

Note that $T = |V| + \sum_{i=1}^{m+2}(e_i - b_i)$ so that under any feasible scheduling solution the machine never idles.

The forbidden and active regions of the machine corresponding to the example of Figure 2 is illustrated in Figure 3. The time limit is $T = 77$.

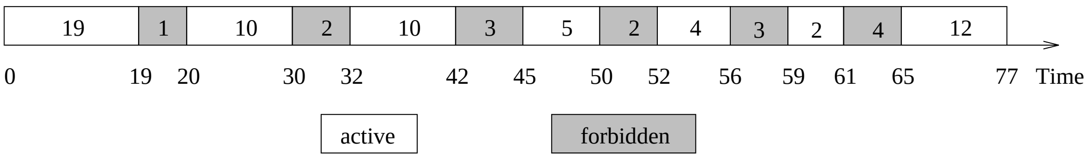

Figure 3: Forbidden and active regions of the machine of the instance (P2).

We claim that *there is a solution to the instance of 3SAT if and only if there there is a feasible schedule for the above instance of (P2).*

The *intuitive idea* behind the proof is that x_i (or $\bar{x}_i$) is true if and only if the execution of x_{i0} (or $\bar{x}_{i0}$) begins in the time interval $[0, m-1]$. The problem instance of (P2) is constructed in such a way that there is a solution to the instance of 3SAT if and only if there is a feasible *non-idle* schedule for the above instance of (P2). In order to have a non-idle solution, we have to schedule in the first $m+1$ active regions the x and y tasks, and in the $m+2$-nd active region the k "clause"-tasks corresponding to truth assignment of literals, and in the last active region the other $6k$ "clause"-tasks. Moreover, in order to have an one-to-one correspondence between the solutions of instances of (P2) and 3SAT, we should schedule in the $m+2$-nd active region k "clause"-tasks corresponding to k different clauses. These k "clause"-tasks are available for execution in the $m+2$-nd active region only if we schedule in the first m time slots either task x_{i0} or $\bar{x}_{i0}$, depending on the truth value associated with each literal x_i being 1 or 0, respectively. The delays on the precedence constraints are chosen such that once all $x_{i,0}$'s and $\bar{x}_{i,0}$'s are executed, we cannot change the order of execution for their successors, i.e. $x_{i,j}$'s and $\bar{x}_{i,j}$'s, for any fixed j, $1 \leq j \leq m$, without introducing at least one idle time in the schedule. These properties are stated and proved in the four claims below.

In the following, we denote by x'_{ij} (resp. y'_i) the first executed task among tasks x_{ij} and $\bar{x}_{ij}$ (resp. y_i and $\bar{y}_i$) and by x''_{ij} (resp. y''_i) the second one. In order to simplify the proof (and the notation in the proof), we consider an additional forbidden region of zero length $F_0 = [b_0, e_0)$ with $b_0 = e_0 = 2m+1$. We have therefore $m+3$ forbidden regions in total (including the above one with zero length): $F_0, F_1, \cdots, F_{m+2}$. Similarly, we consider $m+4$ actives regions $A_0, A_1, \cdots, A_{m+3}$, where A_0 is the active region between time zero and time $2m+1$, and A_1 is the active region from time $2m+1$ to time $4m+3$.

For all tasks $v \in V$, we define $L(v)$, the length of the longest path to vertex v, as follows:

$$L(v) = \begin{cases} \max_{u \in P(v)} L(u) + l_{uv} + 1, & P(v) \neq \emptyset, \\ 0, & P(v) = \emptyset, \end{cases}$$

where $P(v)$ denotes the set of immediate predecessors of v in the task graph.

A task is said to be available at some time t if each of its predecessors has finished execution and the precedence delay between the predecessor to the task has elapsed by time t.

Claim 1: *Tasks with labels x_{ij} or $\bar{x}_{ij}$ which are successors of the first m executed tasks must be executed as soon as they become available in order to have a non-idle schedule.*

Proof of Claim 1:

Tasks x_{im} or $\bar{x}_{im}$ are not available before time $e_{m-1}+1$ due to the fact that

$$L(x_{im}) = L(\bar{x}_{im}) = \frac{5m^2+3m}{2} = e_{m-1}+1.$$

Since the number of time units where the machine is active until time $e_{m-1}+1$ is $2m^2+2m$, which is equal to the maximum number of possibly available tasks until this time $\{x_{ij}, \bar{x}_{ij}, y_{j+1}, \bar{y}_{j+1},\ 1 \le i \le m,\ 0 \le j \le m-1\}$, all these tasks should be executed by time e_{m-1} in order to have a non-idle schedule.

Let $x'_{i_0 0}$ be the task executed at time 0. It is clear that only task $x'_{i_0 m}$ is available at time $e_{m-1}+1$, provided all tasks on the path $x'_{i_0 0}, x'_{i_0 1}, \cdots, x'_{i_0 m}$ are executed as soon as they become available. Let $x_{i_v 0}$ (or $\bar{x}_{i_v 0}$) be the task executed at time v, $1 \le v \le m-1$. The same argument shows that only task $x_{i_v m}$ (or $\bar{x}_{i_v m}$) is available at time $e_{m-1}+1+v$, provided all tasks on the path $x_{i_v 0}, x_{i_v 1}, \cdots, x_{i_v m}$ (or $\bar{x}_{i_v 0}, \bar{x}_{i_v 1}, \cdots, \bar{x}_{i_v m}$) are executed as soon as they become available. □

Claim 2: *In order to have a non-idle schedule, tasks are executed in active region A_0 in the order of*

$$x'_{i_1 0}, x'_{i_2 0}, \cdots, x'_{i_m 0}, x''_{10}, x''_{20}, \cdots, x''_{m0}, y'_1,$$

where $\{i_1, \cdots, i_m\}$ is a permutation on $\{1, 2, \cdots, m\}$, and in active region A_j, $1 \le j \le m-1$, tasks are executed in the order of

$$y''_j, x'_{i_1 j}, x'_{i_2 j}, \cdots, x'_{i_m j}, x''_{1j}, x''_{2j}, \cdots, x''_{mj}, y'_{j+1},$$

and in active region A_m, tasks are executed in the order of

$$y''_m, x'_{i_1 m}, x'_{i_2 m}, \cdots, x'_{i_m m},$$

and in active region A_{m+1}, tasks are executed in the order of

$$x''_{1m}, x''_{2m}, \cdots, x''_{mm}.$$

Proof of Claim 2:

For $1 \le i \le m$, $1 \le j \le m$, the length of the path to tasks x_{ij} and $\bar{x}_{ij}$ is:

$$L(x_{ij}) = L(\bar{x}_{ij}) = (2m+2)j + \frac{j(j-1)}{2} = e_{j-1}+1,$$

and the length of the path to tasks y_j and $\bar{y}_j$ is:

$$L(y_j) = L(\bar{y}_j) = (2m+2)(j-1) + \frac{(j-2)(j-1)}{2} + m + 1.$$

Therefore, the maximum number of possibly available tasks by time $e_{j-1}+1$ is $(2m+2)j$, which is the number of x-tasks with the second index strictly smaller than j and y-tasks with the index smaller than j, $1 \leq j \leq m$. Since the number of time units where the machine is active until time $e_{j-1}+1$ is also equal to $(2m+2)j$, $1 \leq j \leq m$, all these available tasks should be executed by time e_{j-1} in order to have a non-idle schedule.

At time $b_{j-1} - 1 = (2m+2)(j-1) + 2m + \frac{(j-2)(j-1)}{2}$, $1 \leq j \leq m$, tasks x_{ij} or $\bar{x}_{ij}$ are not available, and task y'_j is available if:

$$\sigma(x'_{j0}) + L(y'_j) \leq b_{j-1} - 1,$$

so that

$$\sigma(x'_{j0}) \leq m - 1, \qquad 1 \leq j \leq m.$$

Therefore, tasks $\{x'_{i0}, 1 \leq i \leq m\}$ should be executed in the first m time slots of the schedule in order not to have idle.

Consider now the first time slot of active region A_j, $1 \leq j \leq m$, i.e. time slot $e_{j-1} = b_{j-1} + j - 1$. Task y''_j is available only if

$$\sigma(x''_{j0}) + L(y''_j) \leq e_{j-1},$$

or equivalently,

$$\sigma(x''_{j0}) \leq m + j - 1.$$

An induction on $j = 1, 2, \cdots, m$ yields that task x''_{j0} is scheduled at $m + j - 1$.

Once the schedule of tasks x'_{i0} and x''_{i0}, $1 \leq i \leq m$, are fixed, a simple inductive argument shows that all tasks x_{ij} or $\bar{x}_{ij}$, $j \geq 1$, should be executed as soon as they become available.

It then follows that, in active region A_0, tasks are executed in the order of

$$x'_{i_1 0}, x'_{i_2 0}, \cdots, x'_{i_m 0}, x''_{10}, x''_{20}, \cdots, x''_{m0}, y'_1,$$

where $\{i_1, \cdots, i_m\}$ is a permutation on $\{1, 2, \cdots, m\}$, in active region A_j, $1 \le j \le m-1$, tasks are executed in the order of

$$y''_j, x'_{i_1 j}, x'_{i_2 j}, \cdots, x'_{i_m j}, x''_{1j}, x''_{2j}, \cdots, x''_{mj}, y'_{j+1},$$

and in active region A_m, tasks are executed in the order of

$$y''_m, x'_{i_1 m}, x'_{i_2 m}, \cdots, x'_{i_m m},$$

and in active region A_{m+1}, tasks are executed in the order of

$$x''_{1m}, x''_{2m}, \cdots, x''_{mm}.$$

$\square$

Claim 3 *In order to have a non-idle schedule, only tasks of type c_{rs}, $1 \le r \le k$, $1 \le s \le 7$, are executed in the last two active regions.*

Proof of Claim 3:

The length of the path to tasks of type c_{rs} is

$$L(c_{rs}) = \frac{5m^2 + 7m}{2} + k = b_{m+1}.$$

Hence no task of this type is available before the end of the active region A_{m+1}. Since the number of tasks to be executed in the last two active regions is equal to the number of tasks of type c_{rs}, only tasks c_{rs} are to be executed in the last two active regions in order to have a non-idle schedule. $\square$

Claim 4 *In order to have a non-idle schedule, tasks of type c_{rs}, $1 \le r \le k$, $1 \le s \le 7$, that are available for execution in the active region A_{m+2} should have their predecessors executed in the active region A_m.*

Proof of Claim 4:

The earliest time when some successor of the first executed task of type x''_{jm} in the active region A_{m+1} becomes available is:

$$e_m + 1 + l_{x''_{jm}, c_{rs}} = \frac{5m^2 + 9m}{2} + 2k = b_{m+2}.$$

Thus, in order for some task c_{rs} to be executed in the active region A_{m+2}, all its predecessors, i.e. tasks of type x_{jm} or $\bar{x}_{jm}$, should be executed in the active region A_m. It then follows that the predecessors of tasks that are available for execution in the active region A_{m+2} are of type x'_{jm}. □

Observe that for each pair c_{rs} and c_{rt}, $s \neq t$, there is at least one j such that either x_{jm} precedes c_{rs} and $\bar{x}_{jm}$ precedes c_{rt}, or $\bar{x}_{jm}$ precedes c_{rs} and x_{jm} precedes c_{rt}. Thus, for any r, $1 \leq r \leq k$, one and only one of the seven tasks c_{rs}, $1 \leq s \leq 7$, can be executed in the active region A_{m+2}.

Therefore, we have non-idling scheduling function σ if and only if the first m executed tasks correspond to a satisfying truth assignment. ■

Theorem 1 *Both problems (P1) and (P2) are NP-complete.*

Proof. The assertion follows from Lemmas 1 and 2. ■

Observe that the assertion of the above theorem still holds if the statement of problem (P1) is modified in such a way that the time limit T is bounded: $T \leq 2|V|$. Indeed, in the proof of Lemma 2, the time limit of the corresponding instance of problem (P2) can be bounded by $2|V| - 4$, and in the proof of Lemma 1, the time limit T' of the instance of problem (P1) is $T' = T + 4$, where T is the time limit of the corresponding instance of problem (P2).

Corollary 1 *The problem* $1 \mid prec(l_{ij} \geq 1),\ p_j = 1 \mid C_{\max}$ *is NP-hard.*

Corollary 2 *The problem* $1 \mid prec(l_{ij}),\ p_j = 1 \mid C_{\max}$ *is NP-hard.*

3 Polynomial Solution

In this section, we consider the non-preemptive scheduling problem under the assumption that precedence delays have unit length. However, the task processing times can be arbitrary natural numbers. This problem, denoted by $1 \mid prec(l_{ij} = 1),\ p_j \in I\!N_+ \mid C_{\max}$, will be shown to be polynomial, and we provide an optimal $O(n^2)$ algorithm for the minimization of makespan. At the end of this section we will extend the optimal solution to the case where some of the precedence delays have length zero, and also to the case where release and delivery times are zero or one unit.

The optimal schedule, referred to as Lexicographic Order Schedule (LOS) in this paper, is a list schedule proposed by Coffman and Graham [5] for the makespan minimization of an arbitrary task graph with UET tasks on two parallel processors. LOS is based on a static list of tasks defined by the lexicographic order as follows. Let there be f final tasks. Assign labels $1, \cdots, f$ to these final tasks in an arbitrary way. Suppose now that $k \geq f$ tasks have already been labeled by $1, 2, \ldots k$. Consider all the tasks whose successors are all labeled. Assign label $k+1$ to the task such that the decreasing sequence of the labels of its *immediate* successors is lexicographically minimal (tie is broken in an arbitrary way). LOS is then the list schedule which assigns the available tasks to the machine according to the decreasing order of the labels.

Theorem 2 *Let $G = (V, E)$ be an arbitrary task graph. If precedence delays have unit length, then LOS minimizes the makespan of G within the class of non-preemptive schedules.*

Proof. If the machine does not idle under LOS, then LOS is trivially optimal. Assume in the following that the machine does idle under LOS. Let M be the makespan of G under LOS. Let the label of task $v \in G$ assigned by LOS be $\lambda(v)$. Denote by $\gamma : V \to \{0, 1, 2, \cdots, M-1\}$ the scheduling function of LOS. Denote by $P^*(v)$ the set of all predecessors of v in G.

Consider first the case where all the tasks are UET. For sake of simplicity of notation, we assume, by convention, that whenever the machine is idle before time M, it is executing a *fictitious* task, denoted by 0, with $\lambda(0) = 0$. Note that unless all the tasks have completed execution, the machine never idles two or more units of time contiguously due to the fact that the precedence delays have unit length. Denote by $\gamma^{-1} : \{0, 1, 2, \cdots, M-1\} \to V \cup \{0\}$ the inverse of γ, i.e., $\gamma^{-1}(t)$ denotes the task which is executing on the machine during the time slot $[t, t+1)$ under LOS.

Let s_0 be the earliest time for which the machine is non-idle under LOS during the time interval $[s_0, M)$, i.e. $s_0 - 1$ is the last idle in the schedule:

$$s_0 = \min\left\{s \mid 0 \le s \le M-1,\ \forall t \in \{s, s+1, \cdots, M-1\}: \quad \gamma^{-1}(t) \ne 0\right\}.$$

Denote by V_0 the set of tasks assigned to the machine during the time interval $[s_0, M)$:

$$V_0 = \bigcup_{t=s_0}^{M-1} \{\gamma^{-1}(t)\}.$$

Let $u_0 = 0$ be the *fictitious* task executed at time $s_0 - 1$. Note that the task executed at time $s_0 - 2$ is a predecessor of *all* the tasks in V_0 (otherwise, as precedence delays have unit length, at least one task in V_0 which is not a successor of $\gamma^{-1}(s_0 - 2)$ should be executed at time $s_0 - 1$.) Let s_1 be the earliest time for which the machine is continuously executing predecessors of all the tasks in V_0:

$$s_1 = \min\left\{t \mid 0 \le t \le s_0 - 2, \ \ \forall v \in V_0 : \{\gamma^{-1}(t), \gamma^{-1}(t+1), \cdots, \gamma^{-1}(s_0 - 2)\} \subseteq P^*(v)\right\}.$$

Denote by V_1 the set of tasks assigned to the machine during the time interval $[s_1, s_0 - 1)$:

$$V_1 = \bigcup_{t=s_1}^{s_0-2} \{\gamma^{-1}(t)\}.$$

Let u_1 be the (possibly fictitious) task executed at time $s_1 - 1$. Note that by definition, task u_1 is not a predecessor of *all* the tasks of V_0, whereas tasks of V_1 are the predecessors of all the tasks of V_0. Therefore, according to the definition of the labeling procedure, u_1 has a smaller label than tasks in V_1: $\lambda(u_1) < \min_{v \in V_1} \lambda(v)$. Thus, the task executed at time $s_1 - 2$ is a predecessor of all the tasks in V_1 (otherwise, according to the definition of LOS, at least one task in V_1 which is not a successor of $\gamma^{-1}(s_1 - 2)$ should be executed at time $s_1 - 1$ due to again the assumption that precedence delays have unit length.). Let s_2 be the earliest time for which the machine is continuously executing the predecessors of all the tasks in V_1:

$$s_2 = \min\left\{t \mid 0 \le t \le s_1 - 2, \ \ \forall v \in V_1 : \{\gamma^{-1}(t), \gamma^{-1}(t+1), \cdots, \gamma^{-1}(s_1 - 2)\} \subseteq P^*(v)\right\}.$$

Denote by V_2 the set of tasks assigned to the machine during the time interval $[s_2, s_1 - 1)$:

$$V_2 = \bigcup_{t=s_2}^{s_1-2} \{\gamma^{-1}(t)\}.$$

Let u_2 be the (possibly fictitious) task executed at time $s_2 - 1$. Due to the facts that task u_1 has a smaller label than the tasks in V_1, that task u_2 is not a predecessor of all the tasks of V_1, and that every task in V_2 is predecessor of all the tasks of V_1 and all the tasks of V_0 by transitivity, task u_2 has a smaller label than tasks in V_2: $\lambda(u_2) < \min_{v \in V_2} \lambda(v)$. Thus, the task executed at time $s_2 - 2$ is a predecessor of all the tasks in V_2, so that we can define s_3 as the earliest time for which the machine is continuously executing the predecessors of all the tasks in V_2.

Continue this procedure until the beginning of the schedule, and we obtain the time epochs $0 = s_m < s_{m-1} < \cdots < s_2 < s_1 < s_0 < M$, such that for all $1 \leq i \leq m$, every task of V_i is predecessor of all the tasks of V_{i-1}, where

$$V_i = \bigcup_{t=s_i}^{s_{i-1}-2} \{\gamma^{-1}(t)\}.$$

Let u_i be the (possibly fictitious) task executed at time $s_i - 1$, $0 \leq i \leq m-1$. Such a construction is illustrated in Figure 4.

V_m	u_{m-1}	V_{m-1}	u_{m-2}	$\cdots$	u_2	V_2	u_1	V_1	u_0	V_0

Figure 4: The decomposition of the LOS schedule.

Due to the precedence relations between tasks of V_i and V_{i-1}, $1 \leq i \leq m$, it is clear that any feasible schedule of task graph G has at least length $m + \sum_{i=0}^{m} |V_i|$, where $|V_i|$ denotes the cardinality of V_i. Since $M = m + \sum_{i=0}^{m} |V_i|$, LOS is thus an optimal schedule.

Consider now the general case where task processing times are arbitrary natural numbers. We define the sets of tasks in a similar way, viz., V_0 is the set of tasks executed after the last machine idling, and tasks of V_i, $1 \leq i \leq m$, are consecutively executed, and each task of V_i is predecessor of all the tasks of V_{i-1}. Since precedence delays have unit length, there is only one (possibly fictitious) task, denoted by u_{i-1}, in-between tasks of V_i and those of V_{i-1} in the LOS schedule. Let U be the set of non-fictitious tasks executed in-between sets V_i, $1 \leq i \leq m$:

$$U \stackrel{\text{def}}{=} V - \bigcup_{i=0}^{m} V_i = \bigcup_{i=1}^{m} \{u_i\} - \{0\}.$$

Then, the makespan is equal to the total processing time of G and number of idling slots:

$$M = \sum_{i=1}^{m}(\mathbf{1}_{\{u_i \neq 0\}} p_{u_i} + \mathbf{1}_{\{u_i = 0\}}) + \sum_{i=0}^{m} \sum_{v \in V_i} p_v = W + m - |U|, \tag{1}$$

where $W = \sum_{v \in V} p_v$ is the total processing time of G, and $\mathbf{1}_{\{\bullet\}}$ is the indicator function.

Consider an arbitrary schedule π for G with makespan M'. In the following, we show that makespan M' of π is never less than makespan M of LOS. The basic idea is that any schedule has to finish *all* tasks in V_i before executing some task from V_{i-1}.

Let s'_i (resp. t'_i) be the time epoch when the first (resp. last) task of V_i starts execution under schedule π, $0 \leq i \leq m$. Since every task of V_i, $1 \leq i \leq m$, is predecessor of all the tasks of V_{i-1}, we obtain that $t'_i - 2 \leq s'_{i-1}$. Let V'_i be the set of tasks which start execution under π during time interval $[s'_i, t'_i]$, $0 \leq i \leq m$. Clearly, $V_i \subseteq V'_i$, $0 \leq i \leq m$. Denote by U'_{i-1} the set of (non-fictitious) tasks which start execution under π during time interval (t'_i, s'_{i-1}), $1 \leq i \leq m$. Let U' be the set of non-fictitious tasks executed in-between sets V_i and V_{i-1}, $1 \leq i \leq m$. It is simple that

$$U' \stackrel{\text{def}}{=} \bigcup_{i=1}^{m} U'_i = V - \bigcup_{i=0}^{m} V'_i \subseteq V - \bigcup_{i=0}^{m} V_i = U. \tag{2}$$

Therefore,

$$\begin{aligned} M' &\geq \sum_{i=1}^{m} \left(\mathbf{1}_{\{U'_i \neq \emptyset\}} (\sum_{u \in U'_i} p_u) + \mathbf{1}_{\{U'_i = \emptyset\}} \right) + \sum_{i=0}^{m} \sum_{v \in V'_i} p_v \\ &= W + m - \sum_{i=1}^{m} \mathbf{1}_{\{U'_i \neq \emptyset\}} \\ &\geq W + m - |U'|, \end{aligned} \tag{3}$$

where the first inequality is due to the fact that schedule π may have idling periods.

Inequality (3) together with relations (1) and (2) immediately imply that

$$M' \geq W + m - |U'| \geq W + m - |U| = M.$$

Therefore, LOS has a minimum makespan. ∎

In case of UET tasks, the optimality of LOS remains true even when lengths of precedence delays are allowed to be zero in-between tasks of a subchain of the task graph, i.e., $l_{ij} = 0$ only if $|S(i)| = |P(j)| = 1$, where $S(i)$ denotes the set of immediate successors of i. In this case, tasks i and j are given the same lexicographic-order label.

More specifically, we define a Modified Lexicographic Order Schedule (MLOS), based on the following modified lexicographic-order labeling: Let there be f final tasks. Assign labels $1, \cdots, f$ to these final tasks in an arbitrary way. Suppose now that $k \geq f$ tasks have already been labeled by $1, 2, \ldots k$. Consider all the tasks whose successors are all labeled. If the task whose decreasing sequence of the labels of immediate successors is lexicographically minimal has a unique successor, and if the precedence delay between this task and its successor has length zero, then the task is assigned the same label as its successor. Otherwise, this task is assigned label $k + 1$.

Theorem 3 *Let $G = (V, E)$ be an arbitrary task graph with UET tasks. Assume that for all $(u, v) \in E$, $l_{uv} \in \{0, 1\}$, and that $|S(u)| = |P(v)| = 1$ whenever $l_{uv} = 0$. Then MLOS minimizes the makespan of G within the class of non-preemptive schedules.*

Proof. The proof is analogous to the first part (for the case of UET tasks) of the proof of Theorem 2. We can define time epochs $0 = s_m < s_{m-1} < \cdots < s_2 < s_1 < s_0 < M$, in such a way that for all $1 \leq i \leq m$, every task of V_i is predecessor of all the tasks of V_{i-1}, where

$$V_i = \bigcup_{t=s_i}^{s_{i-1}-2} \{\gamma^{-1}(t)\}.$$

If $l_{uv} = 0$ and $v \in V_i$ for some $0 \leq i \leq m$, then, according to the assumption, $|S(u)| = |P(v)| = 1$, so that $u \in V_i$. Thus, for any $(u, v) \in E$, if $u \in V_{i-1}$ and $v \in V_i$, then $l_{uv} \geq 1$. Therefore, the arguments of the optimality of LOS provided in the proof of Theorem 2 are still valid. ∎

Remark: It is easily seen from the above proof that the optimality of MLOS extends to the case where the processing times of u and v are arbitrary natural numbers whenever $l_{uv} = 0$.

Remark: As mentioned previously, precedence delays can be used to model release and delivery times. Thus, the above result of polynomial solution holds for the case where release and delivery times are unit length. In fact, the polynomial solution can be extended to the case where release and delivery times are zero or unit length. In this case, lexicographic-order labeling starts with the final tasks which have zero delivery times. The proof of the optimality of such an LOS can be carried out by adding a fictive task as the successor of all the final tasks which have unit delivery times. The rest of the proof is analogous to that of Theorem 2.

In LOS and MLOS, the lexicographic-order labeling requires $O(|E|)$ operations, and the on-line scheduling requires $O(n \log n)$ operations. Since $|E| \leq n^2$, the time complexity of LOS is therefore $O(n^2)$.

4 Preemptive Scheduling

In this section, we consider the preemptive case $1 \mid pmtn,\ prec(l_{ij}),\ p_j \mid C_{\max}$. It will be shown in Lemma 3 below that preemptive solutions are not dominant when tasks have UET.

Lemma 3 *Let $G = (V, E)$ be an arbitrary task graph with UET tasks and integer precedence delays. Then for any preemptive schedule S of G, there is a non-preemptive schedule S' of G obtained from a polynomial transformation of S, such that the makespan under S' is the same as the makespan under S.*

Proof. Let M be the makespan of G under schedule S. Assume without loss of generality that under S, the tasks of $V = \{1, 2, \cdots, n\}$ complete execution in the order of $1, 2, \cdots, n$.

Denote by $k_1^1 < k_2^1 < \cdots < k_{l_1}^1$ the tasks executed during the time slot $[0, 1)$ under S. Let $r(k_i^1)$, $1 \leq i \leq l_1$, be the execution time of these tasks in $[0, 1)$. Construct schedule S^1 as follows. In the time slot $[0, 1)$, S^1 executes exclusively task k_1^1. Starting from time epoch 1, S^1 assigns the same tasks to the machine as S does except for task k_1^1. Whenever S assigns task k_1^1 to the machine, S^1 assigns firstly task k_2^1, secondly task k_3^1, etc., and finally task $k_{l_1}^1$, such that among the total amount of $1 - r(k_1^1)$ execution time of task k_1^1, task k_i^1 occupies the machine for a total amount $r(k_i^1)$ of time, $2 \leq i \leq l_1$. It is easy to see that S^1 is a feasible schedule, and that all tasks (in particular, task k_1^1) complete execution earlier (i.e. no later) under S^1 than under S.

Consider now schedule S^1. Denote by $k_1^2 < k_2^2 < \cdots < k_{l_2}^2$ the tasks executed during the time slot $[1,2)$ under S^1. Let $r(k_i^2)$, $1 \le i \le l_2$, be the execution time of these tasks in $[1,2)$. Construct schedule S^2 in the same way as we do for S^1. In the time slot $[1,2)$, S^2 executes exclusively task k_1^2. During time intervals $[0,1)$ and $[2,M)$, S^2 assigns the same tasks to the machine as S^1 does except for task k_1^2. Whenever S^1 assigns task k_1^2 to the machine, S^2 assigns firstly task k_2^2, secondly task k_3^2, etc., and finally task $k_{l_2}^2$, such that among the total amount $1 - r(k_1^2)$ of execution time of task k_1^2, task k_i^2 occupies the machine for a total amount $r(k_i^2)$ of time, $2 \le i \le l_2$. Again, it is easily seen that S^2 is a feasible schedule, and that all tasks (in particular, task k_1^2) complete execution earlier under S^2 than under S^1.

In general, for $2 \le m \le M$, we define schedule S^m based on schedule S^{m-1}. Denote by $k_1^m < k_2^m < \cdots < k_{l_m}^m$ the tasks executed during the time slot $[m-1,m)$ under S^{m-1}. Let $r(k_i^m)$, $1 \le i \le l_m$, be the execution time of these tasks in $[m-1,m)$. Construct schedule S^m as follows. In the time slot $[m-1,m)$, S^m executes exclusively task k_1^m. During time intervals $[0,m-1)$ and $[m,M)$, S^m assigns the same tasks to the machine as S^{m-1} does except for task k_1^m. Whenever S^{m-1} assigns task k_1^m to the machine, S^m assigns firstly task k_2^m, secondly task k_3^m, etc., and finally task $k_{l_m}^m$, such that among the total amount $1 - r(k_1^m)$ of execution time of task k_1^m, task k_i^m occupies the machine for a total amount $r(k_i^m)$ of time, $2 \le i \le l_m$. One easily sees that all tasks (in particular, task k_1^m) complete execution earlier under S^m than under S^{m-1}.

Before proceeding with the proof of the feasibility of the intermediate schedules, we provide an example of the construction. Consider the four tasks and its preemptive schedule S in Figure 5. The tasks complete execution in the order of $1, 2, 3, 4$ in S.

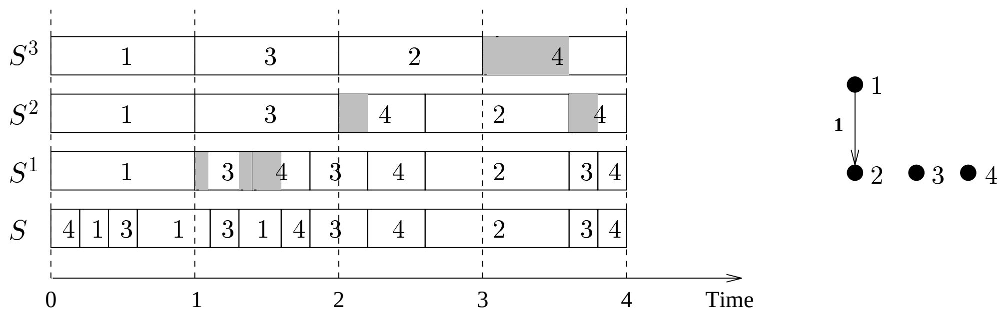

Figure 5: Example of construction of a non-preemptive schedule.

- The preemptive schedule S executes three tasks in the time slot $[0, 1)$, i.e. tasks $k_1^1 = 1, k_2^1 = 3, k_3^1 = 4$. The (fraction of) execution time for the tasks in this slot is $r(1) = 0.6, r(3) = 0.2, r(4) = 0.2$. S^1 is same as S except for the execution of tasks $1, 3, 4$. Task 1 is completed at the end of time interval $[0, 1)$, and the fractions $r(3), r(4)$ of tasks $3, 4$ are executed during the intervals of execution of task 1 after time 1 in S, which are the dashed parts of the execution of tasks $3, 4$ in S^1.

- In schedule S^1 tasks that are executed during the time slot $[1, 2)$ are $k_1^2 = 3, k_2^2 = 4$. The fractions of execution times of tasks 3 and 4 in this slot are $r(3) = 0.6$ and $r(4) = 0.4$. S^2 is same as S^1 except for the execution of tasks $3, 4$. Task 3 is completed at the end of time interval $[1, 2)$, and the fraction $r(4)$ of task 4 is executed during the intervals of execution of task 3 after time 2 in S^1 (dashed region in S^2).

- Finally, to construct S^3, we have the tasks that are executed in S^2 during the third slot that are $k_1^3 = 2, k_2^3 = 4$ with fractions $r(2) = 0.4$, $r(4) = 0.6$. Task 2 will be completed in the third slot in S^3 and obviously task 4 in the fourth slot.

- The last schedule S^4 is identical to the schedule S^3 since in the initial schedule there is no idle time (thus, we omitted S^4 in the figure).

We now prove the feasibility of S^m by induction on m. As we mentioned previously, the feasibility of S^1 is trivial. Assume S^{m-1} is a feasible schedule. In order to prove the feasibility of S^m, it suffices to analyze task k_1^m which is the only task that might start execution strictly earlier in S^m than in S^{m-1}. Note that the following facts hold

- all the precedence delays are integers;

- in schedules S^{m-1} and S^m, all the predecessors of task k_1^m finish execution at integer time epochs no later than time $m - 1$;

- schedule S^{m-1} is feasible;

- task k_1^m is assigned to the machine during the time slot $[m - 1, m)$.

Therefore, in schedules S^{m-1} and S^m, all precedence delays between task k_1^m and its predecessors have elapsed by time $m - 1$. Indeed, if a predecessor of k_1^m, denoted by v, is finished at time $t \leq m-1$ in the feasible schedule S^{m-1}, then $l_{v,k_1^m} \leq s-t$, where $s < m$ is the starting time of k_1^m in S^{m-1}. As l_{v,k_1^m} and t are integers, we have necessarily $l_{v,k_1^m} \leq m-1-t$. Thus task k_1^m is executable at time $m-1$ in schedule S^m.

Consider the final schedule S^M under which all tasks finish execution earlier than under S. Since S^M is a non-preemptive schedule by definition, we can take S^M as S', and the proof is thus completed. ∎

In the above lemma, the precedence delays can be zero. Thus it implies schedules with preemptions occurring at integer instants of time are dominant in the case of integer execution times:

Corollary 3 *Let $G = (V, E)$ be an arbitrary task graph with integer task execution times and integer precedence delays. Then for any preemptive schedule S of G, there is a schedule S' of G, obtained from a polynomial transformation of S, such that S' preempts only at integer time epochs and that the makespan under S' is the same as the makespan under S.*

As a consequence of Lemma 3 and Corollaries 1 and 2, we obtain

Corollary 4 *The problem $1 \mid pmtn,\ prec(l_{ij} \geq 1),\ p_j = 1 \mid C_{\max}$ is NP-hard.*

Corollary 5 *The problem $1 \mid pmtn,\ prec(l_{ij}),\ p_j = 1 \mid C_{\max}$ is NP-hard.*

In view of Lemma 3, in case of preemptive scheduling, one only need to split tasks to UET tasks. Let PLOS denote the Preemptive Lexicographic Order Schedule which splits, if necessary, tasks to UET tasks. In other words, at each integer time epoch, PLOS assigns an executable task to the machine for one unit of time according to the lexicographic-order labeling of the tasks. Applying Lemma 3 and Theorem 3 implies

Theorem 4 *Let $G = (V, E)$ be an arbitrary task graph. If precedence delays have unit length, then PLOS minimizes the makespan of G within the class of preemptive schedules.*

Proof. Let G' be the task graph obtained from replacing each task i of G by a chain of p_i UET tasks. The precedence delays on these chains have length zero. Thus, an application of Lemma 3 and Theorem 3 implies that MLOS minimizes the makespan of G' within the class of preemptive schedules. The assertion of the theorem now follows from the facts that MLOS for G' coincides with PLOS for G, and that the optimal preemptive schedules of G and G' have the same makespan. ∎

5 Conclusions

In this paper, we have considered a single-machine scheduling problem with precedence delays for the minimization of makespan. We have analyzed both preemptive and non-preemptive cases. We have shown that the problem is NP-hard when tasks have unit execution times and precedence delays have integer lengths. We have provided an $O(n^2)$ optimal algorithm when tasks have arbitrary integer execution times and precedence delays have unit length.

Note that the polynomial solution LOS is not optimal for two machines. A counterexample is illustrated in Figure 6, where all the processing times and precedence delays have unit length. The Gantt charts in Figure 6 indicates that an optimal solution has no idle and yields a strictly smaller makespan than that of LOS.

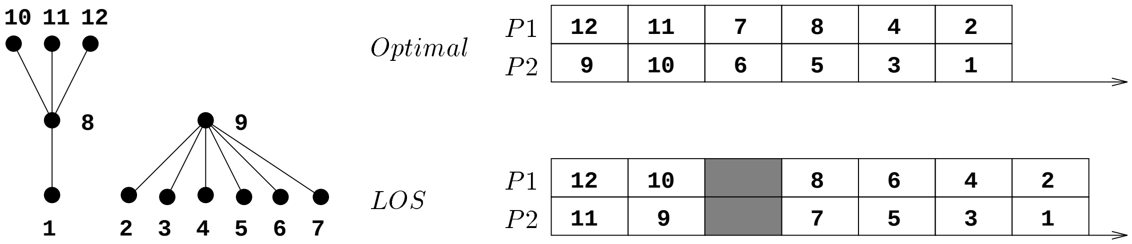

Figure 6: A counterexample of LOS in two machines.

Acknowledgments: The authors are grateful to the referees for useful comments on the first draft of the paper.

References

[1] J. Adams, E. Balas, D. Zawack, "The Shifting Bottleneck Procedure for Job Shop Scheduling", *Management Science*, **34** (1988), pp. 391-401.

[2] K. R. Baker, *Introduction to Sequencing and Scheduling*, J. Wiley, New York, 1974.

[3] E. Balas, J.K. Lenstra, A. Vazacopoulos, "One Machine Scheduling with Delayed Precedence Constraints", CWI Research report BS-R9304, 1993.

[4] J. Carlier, "The One-Machine Scheduling Problem", *E. J. O. R.*, **11** (1982), pp. 42-47.

[5] E. G. Coffman, Jr and R. L. Graham, "Optimal Scheduling for Two-Processor Systems", *Acta Informatica*, **1**, (1972), pp. 200-213.

[6] S. Dauzère-Peres, J. B. Lasserre, "A Modified Shifting Bottleneck Procedure for Job-Shop Scheduling", *Int. J. Prod. Res.*, **31** (1993), pp. 923-932.

[7] S. Dauzère-Peres, J. B. Lasserre, "Integration of lotsizing and scheduling decisions in a job-shop", *E. J. O. R.*, **75** (1994), pp. 413-426.

[8] P. Dileepan, T. Sen, "Bicriterion Static Scheduling Research for a Single Machine", *Omega*, **16** (1988), pp. 53-59.

[9] L. Finta, Z. Liu, "Scheduling of Parallel Programs in Single-Bus Multiprocessor Systems", Rapport de Recherche INRIA, No. 2302, 1994.

[10] M. R. Garey, D.R. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[11] M. R. Garey, D.R. Johnson, B. B. Simons, R. E. Tarjan, "Scheduling Unit-Time Tasks with Arbitrary Release Times and Deadlines", *SIAM J. Comput.*, **10** (1981), pp. 256-269.

[12] R. L. Graham, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, "Optimization and Approximation in Deterministic Scheduling: A Survey", *Ann. Disc. Math.*, **5** (1979), pp. 287-326.

[13] S. K. Gupta, J. Kyparisis, "Single Machine Scheduling Research", *Omega*, **15** (1987), pp. 207-227.

[14] W. A. Horn, “Some Simple Scheduling Algorithms”, *Naval Res. Logist. Quart.*, **21** (1974), pp. 177-185.

[15] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys, “Sequencing and Scheduling: Algorithms and Complexity”, CWI Report BS-R8909, 1989.

[16] J. D. Ullman, “NP-complete Scheduling Problems”, *Journal of Computer and System Sciences*, **10** (1975), pp. 384–393.

[17] B. B. Simons, “A Fast Algorithm for Single Processor Scheduling”, *19-th Annual Symposium on Foundations of Computer Science*, **9** (1978), pp. 246–252.

[18] B. Veltman, B. J. Lageweg, J. K. Lenstra, “Multiprocessor Scheduling with Communication Delays”, *Parallel Computing*, **16** (1990), pp. 173-182.