

# Generalization of EDF and LLF: identifying all optimal online algorithms for minimizing maximum lateness

Patchrawat “Patch” Uthaisombut  
Department of Computer Science, University of Pittsburgh, USA  
utp@cs.pitt.edu, <http://www.cs.pitt.edu/~utp>

May 5, 2005

## Abstract

It is well known that the Earliest-Deadline-First (EDF) and the Least-Laxity-First (LLF) algorithms are optimal algorithms for the problem of preemptively scheduling jobs that arrive over time on a single machine to minimize maximum lateness ( $1|r_j, pmtn|L_{\max}$ ). It was not previously known what other online algorithms are optimal for this problem. As this problem is a fundamental problem in machine scheduling, it deserves a thorough investigation. In this paper, the concept of *compound laxity* is introduced, and a complete characterization of all optimal online algorithms for this problem is derived.

## 1 Introduction

We consider the problem of preemptively scheduling jobs that arrive over time on a single machine to minimize maximum lateness. This problem is denoted  $1|r_j, pmtn|L_{\max}$  according to Graham et. al.’s notation [5]. It is well known that the Earliest-Deadline-First (EDF) algorithm and the Least-Laxity-First (LLF) algorithm are optimal for this problem [4, 3]. EDF always runs an available job with the smallest deadline. LLF always runs an available job with the smallest laxity where the laxity of a job at time  $t$  is defined as the difference between its deadline and the sum of  $t$  and the remaining processing time of the job at time  $t$ . The laxity of a job indicates, on a job-by-job basis, how much the job can be delayed without being late. Both algorithms are online algorithms, which construct a schedule over time without knowledge of the existence of jobs that have not arrived.

Evidently, there are other online algorithms which are optimal for this problem as well. However, an exact characterization of these algorithms was not previously known. This is in sharp contrast with another fundamental problem, the problem of preemptively scheduling jobs that arrive over time on a single machine to minimize the total completion time. This problem is denoted  $1|r_j, pmtn|\sum C_j$  according to Graham et. al.’s notation [5]. The Smallest-Remaining-Processing-Time (SRPT) algorithm is optimal for this problem [1]. SRPT always runs an available job with the smallest remaining processing time. For an algorithm to be optimal for the total completion time problem, it must follow the SRPT rule at any time. Any algorithm that deviates from the SRPT

rule is not optimal. The rule is very rigid. Therefore, in fact, there is only one algorithm that is optimal for the total completion time problem. In contrast, there are at least two distinct optimal algorithms, namely EDF and LLF, and probably many more, for the maximum lateness problem. Evidently, the governing rule that promises optimality for the maximum lateness problem is more flexible than the total completion time problem, and EDF and LLF are merely manifestations of this underlying rule.

In this paper, we introduce the *compound laxity* of jobs. Similar to the *laxity* of a job, the *compound laxity* of a job quantifies how close the job is to being late. Compound laxity leads to the *compound laxity rule* (CL rule), which is a natural rule for selecting a job to run at any time. We give a complete characterization of all optimal online algorithms for the maximum lateness problem; an online algorithm is optimal if and only if it follows the CL rule at all time.

The rest of the paper is organized as follows. In section 2, the definition of the problem and basic quantities are given. The compound laxity and the compound laxity rule are defined. Section 3 furnishes the relationship between compound laxity and lateness. Section 4 shows that an online algorithm is optimal if and only if it always follows the CL rule. Table 1 summarizes the results in sections 3 and 4. Section 5 discusses some related open problems.

## 2 Definitions

### 2.1 Problem Definition

The problem of preemptively scheduling jobs that arrive over time on a single machine to minimize maximum lateness is considered. This problem is denoted  $1|r_j, pmtn|L_{\max}$  according to Graham et. al.'s notation [5]. An input instance in this problem consists of  $n$  jobs. Job  $j$  arrives at time  $r_j$ , has a processing time  $p_j$ , and a due date  $d_j$ . An algorithm for this problem must schedule the jobs preemptively on one machine. Let  $C_j^S$  denote the completion time of job  $j$  in schedule  $S$ . The *lateness* of job  $j$  in schedule  $S$ , denoted  $L_j^S$ , is defined as  $L_j^S = C_j^S - d_j$ . The *maximum lateness* of schedule  $S$ , denoted  $L_{\max}^S$ , is defined as  $L_{\max}^S = \max_j L_j^S$ . Note that the lateness of a job could be negative if it finishes before its deadline, and the maximum lateness of a schedule could be negative if all jobs finish before their deadlines. The goal is to find a schedule with the smallest maximum lateness. For any input instance  $I$ , let  $L_{\max}^*(I)$  denote the maximum lateness in an optimal schedule for  $I$ . An algorithm is *online* if it is not aware of the existence of jobs that have not arrived. When a job arrives, its processing time and due date become known to the online algorithm. An algorithm is *offline* if it is aware of all jobs and their parameters in advance. Let  $A(I)$  denote the schedule produced by algorithm  $A$  for input instance  $I$ .

### 2.2 Basic Definitions

For any non-empty subset  $X$  of jobs in  $I$ , let

- $r(X) = \min_{i \in X} r_i$ , the earliest arrival time of any job in  $X$ ,

- $p(X) = \sum_{i \in X} p_i$ , the total processing time of jobs in  $X$ , and
- $d(X) = \max_{i \in X} d_i$ , the latest due date of any job in  $X$ .

By convention, if a job runs continuously from time  $s$  to time  $t$ , we say that it runs in the close-open interval  $[s, t)$ . For any input instance  $I$ , any schedule  $S$  for  $I$ , any times  $s, t$  where  $r_j \leq s \leq t \leq C_j^S$ , and any non-empty subset  $X$  of jobs in  $I$ , let

- $p_j^S(t)$  be the remaining processing time of job  $j$  at time  $t$  in schedule  $S$ ,
- $q_j^S(t)$  be the amount of work done on job  $j$  by time  $t$  in schedule  $S$ ,
- $p^S(X, t) = \sum_{i \in X} p_i^S(t)$ , the total remaining processing time of jobs in  $X$  at time  $t$ ,
- $q^S(X, t) = \sum_{i \in X} q_i^S(t)$ , the total amount of work done on jobs in  $X$  by time  $t$ ,
- $q_j^S(s, t) = q_j^S(t) - q_j^S(s)$ , the amount of work done on job  $j$  between time  $s$  and time  $t$ , and
- $q^S(X, s, t) = \sum_{i \in X} q_i^S(s, t)$ , the total amount of work done on jobs in  $X$  between time  $s$  and time  $t$ .

Note that  $p_j^S(t) + q_j^S(t) = p_j$  for any time  $t$  where  $r_j \leq t \leq C_j^S$ . Also,  $p^S(X, t) = p^S(X_1, t) + p^S(X_2, t)$  for any partitioning of  $X$  into  $X_1$  and  $X_2$ . Similar equalities hold for  $q^S(X, t)$  and  $q^S(X, s, t)$ . We define more quantities below. These quantities are needed for defining compound laxity and compound laxity algorithms in the next subsection.

- Let  $B_j = \{ i \mid d_i \leq d_j \}$ , the set of jobs with deadlines no later than  $d_j$ . We refine the definition of  $B_j$  in several ways as follows.
- Let  $B_j(t) = \{ i \mid r_i \leq t \text{ and } d_i \leq d_j \}$ , the set of jobs that arrive by time  $t$  with deadlines no later than  $d_j$ . Also, this is the set of jobs with deadlines no later than  $d_j$  seen by an online algorithm at time  $t$ .
- Let  $B_j(s, t) = B_j(t) - B_j(s) = \{ i \mid s < r_i \leq t \text{ and } d_i \leq d_j \}$ , the set of jobs that arrive in the interval  $(s, t]$  with deadlines no later than  $d_j$ .
- Let  $B_{j,k}(t) = B_k(t) - B_j(t) = \{ i \mid r_i \leq t \text{ and } d_i \leq d_j \}$ , the set of jobs that arrive by time  $t$  with deadlines in the interval  $(d_j, d_k]$ .

For example, consider the instance  $I$  in the top part of Figure 1. There are 6 jobs designated by rectangles with different fill patterns. Each job is labeled with a job id. The horizontal lengths of the rectangles represent the processing times. The positions of the left most portion of the rectangles represent the arrival times. The positions of the small vertical bars to the right of the rectangles connected by horizontal lines represent the deadlines. The horizontal lines are redundant and only serve as a visual aid to connect corresponding rectangles and vertical bars. Nevertheless,

the length of a horizontal line is the “laxity” (defined in the next subsection) of the job when it arrives. For this instance,

$$\begin{aligned}
B_5 &= \{1, 2, 3, 5, 6\} & B_5(3) &= \{1, 2, 3\} & B_5(3, 5) &= \{5, 6\} \\
& & B_5(5) &= \{1, 2, 3, 5, 6\} & B_{3,5}(3) &= \{1\} \\
& & B_3(3) &= \{2, 3\} & &
\end{aligned}$$

We note the following. If  $d_i = d_j$ , then  $B_j(t) = B_i(t)$  and  $l_i^S(t) = l_j^S(t)$ . Values of  $B_j$ ,  $B_j(t)$ ,  $B_j(s, t)$ , and  $B_{j,k}(t)$  depend only on the input instance and are independent of the schedule.

## 2.3 Compound Laxity Algorithms

We define *compound laxity* and *compound laxity algorithms* in this section.

- A job  $j$  is *available* at time  $t$  if it has arrived but not completed at time  $t$ , *i.e.*,  $r_j \leq t < C_j^S$ .
- The *laxity* of a job  $j$  at time  $t$  in schedule  $S$  is defined as  $d_j - t - p_j^S(t)$ .
- Let  $l_j^S(t)$  denote the *compound laxity* of a job  $j$  at time  $t$  in schedule  $S$ , defined as  $l_j^S(t) = d_j - t - p^S(B_j(t), t)$ .
- Let  $l_{\min}^S(t)$  denote the *critical compound laxity* at time  $t$  in schedule  $S$ , defined as  $l_{\min}^S(t) = \min_j l_j^S(t)$  where the minimum is taken over the set of available jobs at time  $t$  in schedule  $S$ .
- Let  $I_{\min}^S(t) = \{ j \mid l_j^S(t) = l_{\min}^S(t) \}$ , the set of all jobs whose compound laxity is critical.
- Let  $d_{\text{crit}}^S(t)$  denote the *critical deadline* at time  $t$  in schedule  $S$ , defined as  $d_{\text{crit}}^S(t) = \min_{j \in I_{\min}^S(t)} d_j$ . This is the earliest deadline among jobs with critical compound laxity.

*Laxity* is the basis for defining the Least-Laxity-First (LLF) algorithm. At any time  $t$ , LLF runs an available job  $j$  with the smallest laxity [3]. The laxity of job  $j$  is the amount of time that job  $j$  can be delayed without being late ignoring the existence of other jobs in the system. On the contrary, the *compound laxity* of job  $j$  takes into consideration other jobs in the system. It is the amount of time that job  $j$  can be delayed without being late assuming that all other available jobs with smaller or equal deadlines must be completed before  $j$ .

For example, consider schedule  $S$  in Figure 1.  $S3$  is the partial schedule of  $S$  up to time 3.  $I3$  shows the remaining processing times of available jobs at time 3. Jobs are sorted by their deadlines. Note that job 6 is not in  $I3$  because it has not arrived. The length of each horizontal line in  $I3$  is the *laxity* of each job at time 3.  $CL3$  shows the remaining processing times in a way that highlights *compound laxity*. Each row consists of the remaining time of jobs in  $B_j(3)$  where  $j=3,1,5$ , and 4 from top to bottom. The length of each horizontal line in  $CL3$  is the *compound laxity* of the corresponding job  $j$ . The compound laxity is 5,3,3, and 4 for job  $j=3,1,5$ , and 4, respectively. Jobs 1 and 5 (second and third rows) have the smallest compound laxity. Job 1 has a smaller deadline. Thus, the critical deadline is  $d_1 = 11$ .

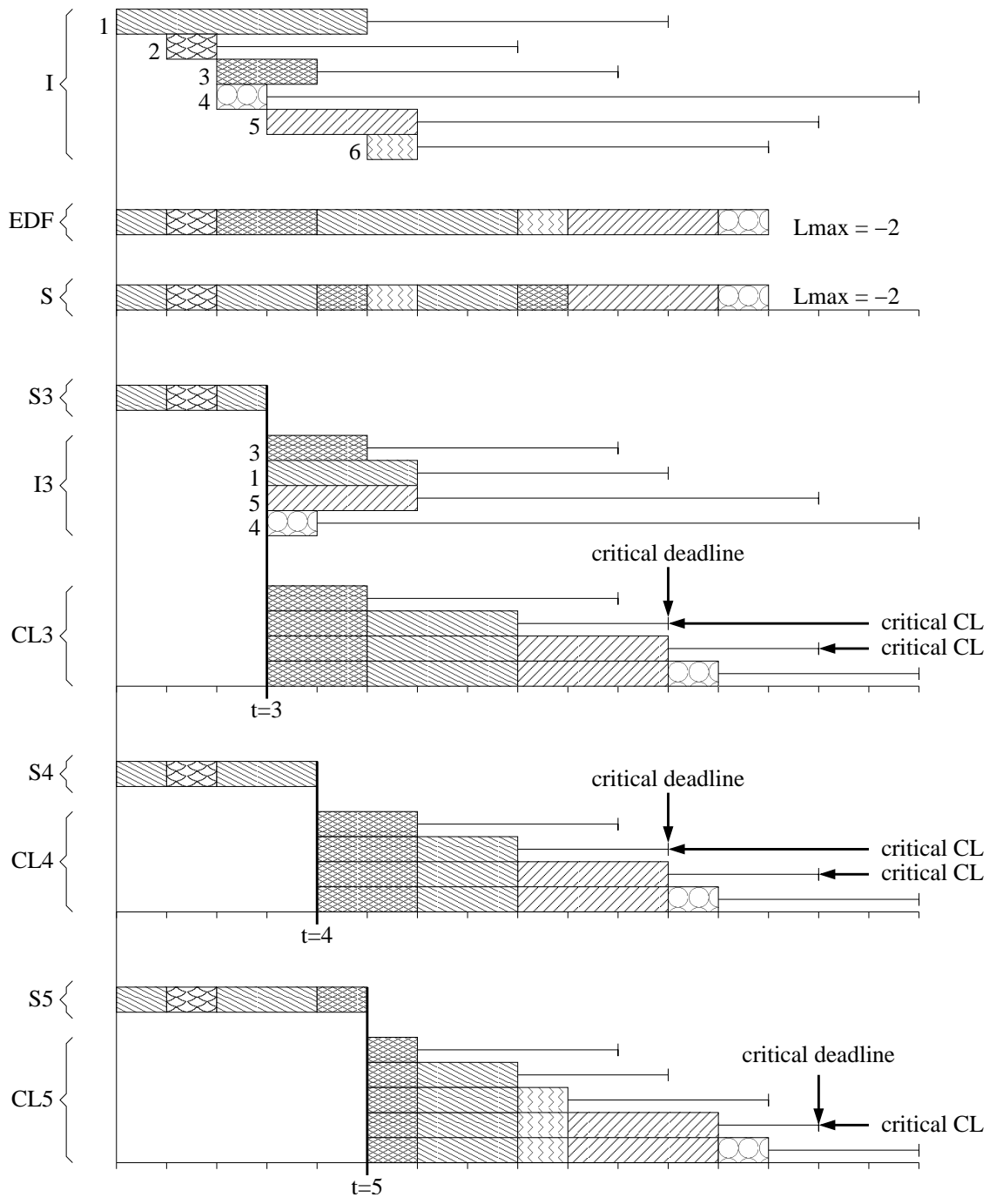


Figure 1: Example illustrating some basic definitions.

Similar to LLF, the idea of *compound laxity rule* is to keep the *critical compound laxity* as large as possible. The smallest deadline of any job with critical compound laxity is called the *critical deadline*. An online algorithm  $A$  is a *compound laxity algorithm* (CL algorithm) if it always runs an available job  $j$  whose deadline is no larger than the critical deadline, that is,  $d_j \leq d_{\text{crit}}^{A(I)}(t)$ . We also say that algorithm  $A$  follows the *compound laxity rule* (CL rule). A schedule  $S$  for an input instance  $I$  is a *compound laxity schedule* (CL schedule) if it can be produced by following the CL rule. The CL rule defines not just one algorithm but a class of algorithms because in general there are several different jobs that can be run at any time.

For example, consider time  $t = 3$  in Figure 1. The critical deadline is  $d_1 = 11$ . To follow the CL rule, either jobs 3 or 1 must be run. In the example, schedule  $S$  runs job 1 in the interval  $[3, 4)$ . Note that the critical deadline does not change in the interval  $[3, 4)$  implying that  $S$  follows the CL rule in this interval. It is left to the reader to verify that  $S$  is a CL schedule, *i.e.* it can be constructed by following the CL rule at all time.

For the rest of this paper, we drop the superscript  $S$  for quantities defined in this section if it is clear from the context.

### 3 Basic Understanding of Compound Laxity

In this section, basic properties of compound laxity and lateness are given. Except for Lemma 3.1, which applies only to optimal schedules, all results in this section apply to any schedule. The results in this section and section 4 are summarized in Table 1. The discussion of the results in this section is given at the end of the section.

Lemmas/Theorems/Corollary	Results
Lemma 3.1	Lower bound of optimal maximum lateness.
Lemma 3.2	Compound laxity of two different jobs.
Lemma 3.3	Compound laxity of a job over time (never increases).
Corollary 3.1	Compound laxity of a job is smallest when it completes.
Lemma 3.4	Compound laxity of a job and its lateness.
Lemma 3.5	Maximum lateness = negation of minimum compound laxity.
Lemma 4.1	Upper bound of maximum lateness of CL algorithms.
Theorem 4.1	CL algorithms are optimal for $1 r_j, pmtn L_{\max}$ .
Lemma 4.2	EDF and LLF are CL algorithms.
Theorem 4.2	Non-CL algorithms are not optimal for $1 r_j, pmtn L_{\max}$ .

Table 1: Summary of results in sections 3 and 4

**Lemma 3.1.** *For any input instance  $I$ ,  $\max_{X \subseteq I} r(X) + p(X) - d(X) \leq L_{\max}^*(I)$  where  $X$  is any non-empty subset of jobs in  $I$ .*

*Proof.* Consider any input instance  $I$ . Let  $S$  be an optimal schedule for  $I$ . Let  $X$  be any non-empty subset of jobs in  $I$ . All jobs in  $X$  start no earlier than  $r(X)$ . The total processing time of jobs in  $X$  is  $p(X)$ . Suppose job  $j$  is the last job in  $X$  to finish in schedule  $S$ . Thus,  $C_j^S \geq r(X) + p(X)$ . The latest due date of jobs in  $X$  is  $d(x)$ . In particular,  $d_j \leq d(X)$ . Thus,  $r(X) + p(X) - d(X) \leq C_j^S - d_j = L_j^S \leq L_{\max}^S = L_{\max}^*(I)$ .  $\square$

**Lemma 3.2.** *For any input instance  $I$ , any schedule  $S$  for  $I$ , any jobs  $i$  and  $j$  in  $I$  such that  $d_i < d_j$ , and any time  $t$  such that  $r_i \leq t \leq C_i^S$  and  $r_j \leq t \leq C_j^S$ , it is the case that  $l_j^S(t) - l_i^S(t) = d_j - d_i - p(B_{i,j}(t), t)$ .*

*Proof.*

$$\begin{aligned} l_j(t) - l_i(t) &= [d_j - t - p(B_j(t), t)] - [d_i - t - p(B_i(t), t)] \\ &= d_j - d_i - [p(B_j(t), t) - p(B_i(t), t)] \\ &= d_j - d_i - p(B_{i,j}(t), t) \end{aligned}$$

$\square$

**Lemma 3.3.** *For any input instance  $I$ , any schedule  $S$  for  $I$ , any job  $j$  in  $I$ , and any time  $s$  and  $t$  such that  $r_j \leq s < t \leq C_j^S$ , it is the case that  $l_j^S(t) = l_j^S(s) - p(B_j(s, t)) - (t - s - q^S(B_j(t), s, t))$ .*

Lemma 3.3 states that the compound laxity of a job at time  $t$  will decrease from that at time  $s$  by the amount that is equal to the processing time of jobs in  $B_j$  that arrive during the interval  $[s, t)$  plus the amount of time during  $[s, t)$  that is NOT spent on jobs in  $B_j$ .

*Proof.* First, note that

$$\begin{aligned} l_j(s) - l_j(t) &= [d_j - s - p(B_j(s), s)] - [d_j - t - p(B_j(t), t)] \\ &= (t - s) + [p(B_j(t), t) - p(B_j(s), s)] \end{aligned}$$

The term  $p(B_j(t), t) - p(B_j(s), s)$  is the difference between the remaining work of jobs in  $B_j(t)$  at time  $t$  and that of jobs in  $B_j(s)$  at time  $s$ . This is equal to  $p(B_j(s, t)) - q(B_j(t), s, t)$ , the processing time of jobs in  $B_j$  that arrive in the interval  $(s, t]$  less the amount of work done on jobs in  $B_j$  in the interval  $(s, t]$ . To show this formally,

$$\begin{aligned} & p(B_j(t), t) - p(B_j(s), s) \\ &= p(B_j(s), t) + p(B_j(s, t), t) - p(B_j(s), s) \quad \text{because } B_j(s, t) = B_j(t) - B_j(s) \\ &= p(B_j(s, t), t) - q(B_j(s), s, t) \quad \text{by definition of } q(X, s, t) \\ &= p(B_j(s, t)) - q(B_j(s, t), t) - q(B_j(s), s, t) \quad \text{because } p(X) = p(X, t) + q(X, t) \\ &= p(B_j(s, t)) - q(B_j(s, t), s) - q(B_j(s, t), s, t) - q(B_j(s), s, t) \quad q(X, t) = q(X, s) + q(X, s, t) \\ &= p(B_j(s, t)) - 0 - q(B_j(s, t), s, t) - q(B_j(s), s, t) \quad \text{because no jobs in } B_j(s, t) \text{ arrive by time } s \\ &= p(B_j(s, t)) - q(B_j(t), s, t) \quad \text{because } B_j(s, t) = B_j(t) - B_j(s) \end{aligned}$$

Thus,  $l_j(s) - l_j(t) = (t - s) + p(B_j(t), t) - p(B_j(s), s) = (t - s) + p(B_j(s, t)) - q(B_j(t), s, t)$ .  $\square$

**Corollary 3.1.** *For any input instance  $I$ , any schedule  $S$  for  $I$ , and any job  $j$  in  $I$ , it is the case that  $\min_{t \in [r_j, C_j^S]} l_j^S(t) = l_j^S(C_j^S)$ .*

*Proof.* Since the machine can spend at most  $t - s$  time units on jobs in  $B_j(t)$  during  $[s, t)$ , then  $q(B_j(t), s, t) \leq t - s$ . Thus, from Lemma 3.3, for times  $s$  and  $t$  such that  $r_j \leq s < t \leq C_j$ , it is the case that  $l_j(t) = l_j(s) - p(B_j(s, t)) - (t - s - q(B_j(t), s, t)) \leq l_j(s)$ . In other words,  $l_j(\cdot)$  can never increase over time. Since  $C_j$  is the latest time in the interval  $[r_j, C_j)$ , the result follows.  $\square$

**Lemma 3.4.** *Consider any input instance  $I$ , any schedule  $S$  for  $I$ , and any job  $j$  in  $I$ .*

(a).  $L_j^S = -l_j^S(C_j^S) - p(B_j(C_j^S), C_j^S) \leq -l_j^S(C_j^S)$ .

(b).  $L_j^S = -l_j^S(C_j^S)$  if and only if job  $j$  is the last job in  $B_j(C_j^S)$  to complete.

(c). If  $L_j^S < -l_j^S(C_j^S)$ , then  $L_x^S \geq -l_j^S(C_j^S) + (d_j - d_x) \geq -l_j^S(C_j^S)$  where  $x$  is the last job in  $B_j(C_j^S)$  to complete.

*Proof.* For part (a),  $L_j = C_j - d_j = -(d_j - C_j - p(B_j(C_j), C_j)) - p(B_j(C_j), C_j) = -l_j(C_j) - p(B_j(C_j), C_j)$ . Next we show part (b). Job  $j$  is the last job in  $B_j(C_j)$  to complete if and only if  $p(B_j(C_j), C_j) = 0$ . This follows directly from definition of  $p(X, t)$ . Thus,  $L_j = -l_j(C_j)$  if and only if job  $j$  is the last job in  $B_j(C_j)$  to complete. Finally, we show part (c). If  $L_j < -l_j(C_j)$ , then job  $j$  is not the last job in  $B_j(C_j)$  to complete. Then  $p(B_j(C_j), C_j) > 0$ . Let  $x$  be the last job in  $B_j(C_j)$  to complete. Thus,  $C_x \geq C_j + p(B_j(C_j), C_j)$ . From definition of  $B_j(C_j)$ , it is the case that  $d_x \leq d_j$ . Thus,  $L_x = C_x - d_x \geq C_j + p(B_j(C_j), C_j) - d_j + (d_j - d_x) = -l_j(C_j) + (d_j - d_x) \geq -l_j(C_j)$ .  $\square$

**Lemma 3.5.** *For any input instance  $I$ , any schedule  $S$  for  $I$ , it is the case that  $L_{\max}^S = -\min_i l_i^S(C_i^S)$ . Further more, for any job  $k$ , it is the case that  $L_k^S = L_{\max}^S$  if and only if  $l_k^S(C_k^S) = \min_i l_i^S(C_i^S)$  and  $k$  is the last job to finish in  $B_k(C_k^S)$ .*

*Proof.* Let  $k$  be any job such that  $L_k = L_{\max}$ . First, we show that  $L_k \geq -\min_i l_i(C_i)$ . Assume to the contrary that  $L_k < -\min_i l_i(C_i)$ . Let  $j$  be a job such that  $l_j(C_j) = \min_i l_i(C_i)$ . Thus,  $L_j \leq L_{\max} = L_k < -\min_i l_i(C_i) = -l_j(C_j)$ . Thus, from Lemma 3.4 part (c), there exists a job  $x$  such that  $L_x \geq -l_j(C_j)$ . Thus,  $L_{\max} < -l_j(C_j) \leq L_x$ . A contradiction. From Lemma 3.4 part (a), it is the case that  $L_k \leq -l_k(C_k)$ . Thus,  $L_k \leq -l_k(C_k) \leq -\min_i l_i(C_i) \leq L_k$ . Hence,  $L_{\max} = L_k = -l_k(C_k) = -\min_i l_i(C_i)$ . From Lemma 3.4 part (b), job  $k$  is the last job in  $B_k(C_k)$  to finish. This proves the first statement of the lemma and the ‘‘only if’’ part of the second statement.

To prove the ‘‘if’’ part of the second statement, assume that  $l_k(C_k) = \min_i l_i(C_i)$  and  $k$  is the last job to finish in  $B_k(C_k)$ . From Lemma 3.4 part (b),  $L_k = -l_k(C_k)$ . From the first statement of the lemma,  $-\min_i l_i(C_i) = L_{\max}$ . Thus,  $L_k = -l_k(C_k) = -\min_i l_i(C_i) = L_{\max}$ .  $\square$

Let us now draw some conclusions from the results in this section and develop a general rule to minimize maximum lateness. Lemma 3.5 tells us that the maximum lateness is equal to the negation of the minimum of the compound laxity of any job when it completes. Lemma 3.3 and Corollary 3.1 tell us that the compound laxity of a job never increases during its life time and is

smallest when it completes. Therefore, to keep the maximum lateness small, an algorithm needs to keep the minimum compound laxity as large as possible.

Next, we discuss how to keep compound laxity large. From Lemma 3.3, between time  $s$  and  $t$ , the compound laxity of job  $j$  *decreases* by  $p(B_j(s, t)) + (t - s - q^S(B_j(t), s, t))$  where  $s < t$ . Suppose no jobs arrive in the interval  $(s, t]$ . This implies  $p(B_j(s, t)) = 0$ . If the machine is devoted to jobs in  $B_j$  during the interval  $(s, t]$ , then the compound laxity of job  $j$  remains the same, which is as large as it could be, throughout the interval  $(s, t]$ . Otherwise, the compound laxity of  $j$  will decrease. If some jobs arrive in the interval  $(s, t]$ , the best an algorithm can do is still to devote the machine to jobs in  $B_j$  for the entire period.

We argue that the development so far leads to the *compound laxity rule*. Observe that EDF always runs an available job with the smallest deadline. Thus, EDF always keeps the compound laxity of *every job* as large as possible at all time. However, this rule can be relaxed. Recall that we only need to keep the minimum compound laxity as large as possible. The minimum compound laxity at time  $t$  is denoted by  $l_{\min}^S(t)$ . The set of all jobs with minimum compound laxity is denoted by  $I_{\min}^S(t)$ . We need to keep the laxity of all such jobs as large as possible. As discussed earlier, to do this, an algorithm should run a job with a deadline smaller than those of jobs in  $I_{\min}^S(t)$ . In other words, the algorithm can run any job  $i$  with  $d_i \leq d_{\text{crit}}^S(t)$  where  $d_{\text{crit}}^S(t)$  is the smallest deadline among jobs in  $I_{\min}^S(t)$ . This is exactly the *compound laxity rule* stated in section 2.3.

## 4 Results on Compound Laxity Algorithms

In this section, results on CL algorithms are given. Lemma 4.1 is the main lemma, which establishes a relationship similar to that in Lemma 3.1. Lemma 3.1 and Lemma 4.1 implies Theorem 4.1, which says that CL algorithms are optimal. In Lemma 4.2, EDF and LLF are shown to be CL algorithms. This implies that EDF and LLF are optimal as stated in Corollary 4.1. Corollary 4.2 states a basic property of EDF, which is used in Theorem 4.2 in showing that non-CL algorithms are not optimal.

**Lemma 4.1.** *For any input instance  $I$  and any CL algorithm  $A$ ,  $L_{\max}^{A(I)} \leq \max_{X \subseteq I} r(X) + p(X) - d(X)$  where  $X$  is any non-empty subset of jobs in  $I$ .*

*Proof.* Fix a CL algorithm  $A$ . Consider any input instance  $I$ . Suppose  $S$  is the schedule produced by  $A$  for  $I$ . Let  $k$  be a job such that  $L_k = L_{\max}$ . On a tie, choose one with the largest deadline. Note, if there are still more than one such jobs, at most one of them will have a non-zero processing time, and all of them complete at the same time. Consider the job set  $B_k(C_k)$ . It is the set of jobs  $i$  such that  $r_i \leq C_k$  and  $d_i \leq d_k$ .

Let  $t$  be the smallest time such that the machine continuously and exclusively runs jobs in  $B_k(C_k)$  in the interval  $[t, C_k)$  in schedule  $S$ . Note that before time  $t$ , the machine is either idle or is running a job not in  $B_k(C_k)$ . Also,  $t$  could possibly be 0 if the machine has been running jobs in  $B_k(C_k)$  since time 0. We will show that  $t < C_k$ . Job  $k$  completes at time  $C_k$ . There must exist some time  $t'$  such that  $t' < C_k$  and the machine continuously processes job  $k$  in the interval  $[t', C_k)$ . Obviously,  $k \in B_k(C_k)$ . Thus,  $t \leq t' < C_k$ .

Let  $Y$  be the set of jobs  $i$  in  $B_k(C_k)$  such that  $t \leq r_i$ , that is,  $i \in Y$  if and only if  $t \leq r_i \leq C_k$  and  $d_i \leq d_k$ . Note that  $Y$  may not be equal to  $B_k(t, C_k)$  as the latter does not include jobs released exactly at time  $t$ .

We claim that during  $[t, C_k)$  the machine continuously and exclusively processes jobs in  $Y$ . This implies that  $k \in Y$ . Thus,  $Y$  is non-empty. From the definition of  $Y$ , it follows that  $r(Y) \geq t$  and  $d(Y) \leq d_k$ . The former and the claim imply that  $p(Y) \geq C_k - r(Y)$ . Hence,  $L_{\max} = L_k = C_k - d_k \leq r(Y) + p(Y) - d(Y) \leq \max_{X \subseteq I} r(X) + p(X) - d(X)$ , which is the statement of the lemma.

It remains to show that during  $[t, C_k)$  the machine continuously and exclusively processes jobs in  $Y$ . In other words, if  $A$  runs a job  $i$  (for a non-zero amount of time) in the interval  $[t, C_k)$ , then  $t \leq r_i \leq C_k$  and  $d_i \leq d_k$ . From the definition of  $t$  and  $B_k(C_k)$ , it is the case that  $d_i \leq d_k$  and  $r_i \leq C_k$ . In the rest of the proof, we will show that  $r_i \geq t$ .

Assume to reach a contradiction that there are jobs  $i$  in  $B_k(C_k)$  that run in the interval  $[t, C_k)$  and  $r_i < t$ . Let  $u$  be a job with the largest deadline among such jobs. Consider time interval  $[r_u, t)$ . Since algorithm  $A$ , which is a CL algorithm, does not insert idle time unnecessarily, then the machine is busy during  $[r_u, t)$ . Since  $u \in B_k(C_k)$ , then

$$d_u \leq d_k \quad (1)$$

Let  $v$  and  $s$  be the job and the smallest time, respectively, such that algorithm  $A$  runs job  $v$  continuously during  $[s, t)$  and no jobs arrive during  $(s, t)$ . It must be the case that  $d_k < d_v$ . Otherwise, this will contradict the choice of  $t$ .

Let  $w$  be a job such that  $d_w = d_{\text{crit}}(s)$ . It is the case that  $d_v \leq d_w$  because algorithm  $A$  follows the CL rule. Thus,  $d_u \leq d_k < d_v \leq d_w$ . It must be the case that  $C_k < C_w$  because job  $w$  is not completed by time  $s$  and it never runs in the interval  $[s, C_k)$ .

Next, we show that  $l_w(s) \leq l_u(s) - (t - s)$ . Assume to the contrary that  $l_w(s) = l_u(s) - \Delta$  for some  $\Delta$  where  $0 \leq \Delta < t - s$ . It is the case that  $l_u(s + \Delta) = l_u(s) - \Delta = l_w(s) = l_w(s + \Delta)$  where the first and the last equalities follows from Lemma 3.3, that algorithm  $A$  runs job  $v$  continuously in the interval  $[s, s + \Delta)$ , and that no jobs arrive during  $(s, t)$ . With a similar argument, for any job  $i$  such that  $d_v \leq d_i$ , it follows that  $l_i(s + \Delta) = l_i(s) \geq l_w(s) = l_w(s + \Delta)$ . We can show that  $d_{\text{crit}}(s + \Delta) < d_v$ . If  $d_v \leq d_{\text{crit}}(s + \Delta)$ , then  $l_{\min}(s + \Delta) = l_i(s + \Delta)$  for some job  $i$  with  $d_v \leq d_i$ . However,  $l_i(s + \Delta) \geq l_w(s + \Delta) = l_u(s + \Delta)$ . Thus,  $l_u(s + \Delta) = l_{\min}(s + \Delta)$  which contradicts that  $d_u < d_v \leq d_{\text{crit}}(s + \Delta)$ . If  $d_{\text{crit}}(s + \Delta) < d_v$ , then job  $v$  should not be chosen to run at time  $s + \Delta$ . A contradiction to that  $v$  runs continuously in the interval  $[s, t)$ . Therefore,

$$d_w - d_u \leq p(B_{u,w}(s), s) - (t - s) = p(B_{u,w(s)}, t) \quad (2)$$

where the inequality follows from Lemma 3.2 (with  $i = u, j = w$  and  $t = s$ ) and that  $l_w(s) - l_u(s) \leq -(t - s)$ , and the equality follows from the fact that the machine continuously processes job  $v$  during  $[s, t)$  and  $v \in B_{u,w}(s)$ .

We claim that all jobs in  $B_{u,k}(s)$  complete by time  $s$ , i.e.  $p(B_{u,k}(s), s) = 0$ . Assume to the contrary that there is a job  $x$  with  $r_x \leq s$ ,  $d_u < d_x \leq d_k$ , and  $p_x(s) > 0$ . Since  $L_k = L_{\max}$ , then from Lemma 3.5,  $L_k = -l_k(C_k)$ , and from Lemma 3.4 part (b), job  $k$  is the last job in  $B_k(C_k)$

to complete. This implies that job  $x$  must complete by time  $C_k$ . Job  $x$  cannot run during  $[s, t)$  because  $v$  is running in that interval. Thus,  $x$  must run for some time in the interval  $[t, C_k)$ . This contradicts the choice of  $u$  because  $d_u < d_x$ . Thus,  $x$  does not exist, and  $p(B_{u,k}(s), s) = 0$  as claimed.

$$\begin{aligned}
p(B_{u,w}(s), t) &= p(B_{u,k}(s), t) + p(B_{k,w}(s), t) \\
&= p(B_{k,w}(s), t) \quad \text{because } 0 \leq p(B_{u,k}(s), t) \leq p(B_{u,k}(s), s) = 0 \\
&= p(B_{k,w}(s), C_k) \quad \text{because jobs } i \text{ with } d_i > d_k \text{ do not run during } [t, C_k) \\
&\leq p(B_{k,w}(C_k), C_k) \quad \text{because } s < C_k
\end{aligned} \tag{3}$$

Thus, from inequalities (1), (2), and (3), it follows that  $d_w - d_k \leq d_w - d_u \leq p(B_{u,w}(s), t) \leq p(B_{k,w}(C_k), C_k)$ . Thus, from Lemma 3.2 (with  $i = k$ ,  $j = w$ , and  $t = C_k$ ), it follows that  $l_w(C_k) \leq l_k(C_k)$ . Then  $-l_w(C_w) \geq -l_w(C_k) \geq -l_k(C_k) = -\min_i l_i(C_i) \geq -l_w(C_w)$  where the first inequality follows from Corollary 3.1, and the equality follows from Lemma 3.5. Thus, all of these quantities are equal. However,  $d_k < d_w$ , which contradicts the choice of  $k$ . Thus, the earlier assumption that  $u$  exists is false, and this completes the proof.  $\square$

**Theorem 4.1.** *For any input instance  $I$  and any CL algorithm  $A$ , it is the case that  $L_{\max}^{A(I)} = \max_{X \subseteq I} r(X) + p(X) - d(X) = L_{\max}^*(I)$ . In other words, any online algorithm  $A$  which is a CL algorithm is optimal for the problem  $1|r_j, pmtn|L_{\max}$ .*

*Proof.* This follows from Lemma 3.1, Lemma 4.1, and the fact that  $L_{\max}^*(I) \leq L_{\max}^{A(I)}$ .  $\square$

**Lemma 4.2.** *Algorithms EDF and LLF are CL algorithms.*

*Proof.* EDF always runs a job with the smallest deadline, which is no larger than the critical deadline. Thus, EDF always follows the CL rule.

Now we show that LLF is a CL algorithm. Suppose to the contrary that there exist an input instance  $I$  and time  $t$  such that LLF violates the CL rule, that is, LLF chooses to run a job with a deadline larger than the critical deadline. Let  $j$  be the job that LLF chooses to run at time  $t$ . Thus,  $d_j - t - p_j^{\text{LLF}}(t) = \min_i d_i - t - p_i^{\text{LLF}}(t)$ . Let  $k$  be a job such that  $l_k^{\text{LLF}}(t) = l_{\min}^{\text{LLF}}(t)$ . Since LLF violates the CL rule at time  $t$ , then  $d_k < d_j$ .

$$\begin{aligned}
& d_j - t - p_j^{\text{LLF}}(t) \leq d_k - t - p_k^{\text{LLF}}(t) \\
\iff & d_j - d_k \leq p_j^{\text{LLF}}(t) - p_k^{\text{LLF}}(t)
\end{aligned} \tag{4}$$

$$\begin{aligned}
& l_k^{\text{LLF}} \leq l_j^{\text{LLF}}(t) \\
\iff & d_k - t - p^{\text{LLF}}(B_k(t), t) \leq d_j - t - p^{\text{LLF}}(B_j(t), t) \\
\iff & d_k - p^{\text{LLF}}(B_k(t), t) \leq d_j - p^{\text{LLF}}(B_j(t), t) \\
\iff & d_j - d_k \geq p^{\text{LLF}}(B_j(t), t) - p^{\text{LLF}}(B_k(t), t) \\
\iff & d_j - d_k \geq p^{\text{LLF}}(B_{k,j}(t), t)
\end{aligned} \tag{5}$$

$$\begin{aligned}
p_j^{\text{LLF}}(t) - p_k^{\text{LLF}}(t) & \geq p^{\text{LLF}}(B_{k,j}, t) \quad \text{from (4) and (5)} \\
& \geq p_j^{\text{LLF}}(t) \quad \text{because } r_j \leq t \text{ and } d_k \leq d_j \\
& > p_j^{\text{LLF}}(t) - p_k^{\text{LLF}}(t) \quad \text{because job } k \text{ has not finished by time } t.
\end{aligned}$$

A contradiction. □

**Corollary 4.1.** *EDF and LLF are optimal algorithms for the problem  $1|r_j, pmtn|L_{\max}$ .*

*Proof.* The results follow from Lemma 4.2 and Theorem 4.1. □

**Corollary 4.2.** *For any input instance  $I$ , any job  $j$  in  $I$ , and any time  $s$  and  $t$  such that  $r_j \leq s < t \leq C_j^{\text{EDF}}$ , if no jobs arrive during  $(s, t]$ , then  $l_j^{\text{EDF}}(t) = l_j^{\text{EDF}}(s)$ .*

*Proof.* Consider any job  $j$ .

$$\begin{aligned}
l_j(t) &= l_j(s) - p(B_j(s, t)) - (t - s - q(B_j(t), s, t)) \quad \text{from Lemma 3.3} \\
&= l_j(s) - (t - s - q(B_j(t), s, t)) \quad \text{because no jobs arrive during } (s, t] \\
&= l_j(s) - (t - s - (t - s)) \quad \text{because EDF always runs a job with} \\
&\quad \text{the smallest deadline (no larger than } d_j), \text{ which must be in } B_j(t) \\
&= l_j(s)
\end{aligned}$$

□

**Theorem 4.2.** *If an online algorithm  $A$  is not a CL algorithm, it is not optimal for the problem  $1|r_j, pmtn|L_{\max}$ .*

*Proof.* Let  $A$  be an online algorithm which is not a CL algorithm. Then there exists an input instance  $I$  and the smallest time  $t$  such that  $A$  does not run any job in  $B_w$  in the interval  $[t, t + \Delta)$  where  $\Delta > 0$  and  $w$  is a job such that  $l_w^{A(I)}(t) = l_{\min}^{A(I)}(t)$ . In other words, algorithm  $A$  violates the

CL rule during the interval  $[t, t + \Delta)$ . Without loss of generality, assume that no jobs arrive during the interval  $(t, t + \Delta)$ . If this is not true, decrease  $\Delta$  until this becomes true.

Let  $I_t$  be the input instance obtained from  $I$  by removing all jobs  $i$  with release time  $r_i \geq t + \Delta$ . Since the online algorithm  $A$  cannot distinguish between  $I$  and  $I_t$  before time  $t + \Delta$ , then the schedule produced by  $A$  for  $I_t$  will be the same as that for  $I$  during  $[0, t + \Delta)$ . Create an instance  $I'$  from  $I_t$  by adding to  $I_t$  a new job  $x$  with

$$r_x = t + \Delta, \quad p_x = \max\{ 0, l_w^{A(I)}(t) + L_{\max}^*(I_t) \}, \quad \text{and} \quad d_x = d_w.$$

We will show that  $L_{\max}^*(I') = \max\{L_{\max}^*(I_t), -l_w^{A(I)}(t)\}$  but  $L_{\max}^{A(I')} \geq \Delta + \max\{L_{\max}^*(I_t), -l_w^{A(I)}(t)\}$ .

First, we show that  $L_{\max}^*(I') = \max\{L_{\max}^*(I_t), -l_w^{A(I)}(t)\}$ . We can construct an optimal schedule for  $I'$  in the following way; use the schedule produced by  $A$  for  $I$  (and the same for  $I_t$ ) for the interval  $[0, t)$ . From time  $t$  on, use EDF. Note that since  $A$  follows the CL rule in the interval  $[0, t)$ , and EDF is a CL algorithm, then the entire schedule is a CL schedule, which is optimal. Call this schedule  $S$ . Note that  $l_w^{A(I')} (t) = l_w^S(t) = l_w^{A(I)}(t)$  because input instances  $I$  and  $I'$  are indistinguishable at time  $t$  and schedules  $A(I)$ ,  $A(I')$ , and  $S$  are the same in the interval  $[0, t)$ . Let  $X = \{ i \mid t \leq r_i \text{ and } C_i \leq t + \Delta \}$ . Let  $Y = \{ i \mid t \leq r_i \text{ and } C_i \geq t + \Delta \}$ . Note that  $X \cup Y \cup \{x\} = I'$ .

For any job  $j \in X$ , it is the case that  $L_j^S \leq L_{\max}^*(I_t)$  because  $X$  is a subset of  $I_t$  and  $S$  looks like an optimal schedule for  $I_t$  in the interval  $[0, t)$ . For any job  $j \in Y$ ,

$$\begin{aligned} L_j^S &\leq -l_j^S(C_j^S) \quad \text{from Lemma 3.4 part (a)} \\ &= -l_j^S(t + \Delta) \quad \text{from Corollary 4.2 and that no jobs arrive after time } t + \Delta \\ &= \begin{cases} -l_j^S(t) + p_x & \text{if } d_x \leq d_j \\ -l_j^S(t) & \text{if } d_j < d_x \end{cases} \\ &\quad \text{from Lemma 3.3, that EDF is the algorithm used, job } x \text{ arrives} \\ &\quad \text{at time } t + \Delta, \text{ and no other jobs arrive in the interval } (t, t + \Delta] \\ &\leq p_x - l_w^S(t) \quad \text{because } l_w^S(t) = l_{\min}^S(t) \leq l_j^S(t) \\ &= \max\{ L_{\max}^*(I_t), -l_w^{A(I)}(t) \} \quad \text{by definition of } p_x \end{aligned}$$

Finally,  $L_x^S \leq -l_x^S(C_x^S) = -l_x^S(t + \Delta) = -l_w^S(t + \Delta)$  where the last equality follows from the definition of compound laxity and that  $d_x = d_w$ . Thus, similar to the derivation above,  $L_x^S \leq -l_w^S(t + \Delta) = \max\{ L_{\max}^*(I_t), -l_w^{A(I)}(t) \}$ . Thus,  $L_{\max}^S = \max_{i \in X \cup Y \cup \{x\}} L_i^S \leq \max\{L_{\max}^*(I_t), -l_w^{A(I)}(t)\}$ .

Next, we show that  $L_{\max}^{A(I')} \geq \max\{L_{\max}^*(I_t), -l_w^{A(I)}(t)\} + \Delta$ .

$$\begin{aligned}
L_{\max}^{A(I')} &= -\min_i l_i^{A(I')}(C_i^{A(I')}) \quad \text{from Lemma 3.5} \\
&\geq -l_w^{A(I')}(C_w^{A(I')}) \\
&\geq -l_w^{A(I')}(t + \Delta) \quad \text{from Corollary 3.1} \\
&= -l_w^{A(I')}(t) + p_x + \Delta \quad \text{from Lemma 3.3, arrival of job } x, \text{ and that} \\
&\quad \text{jobs in } B_w \text{ do not run during } [t, t + \Delta) \\
&= \max\{L_{\max}^*(I_t), -l_w^{A(I)}(t)\} + \Delta \quad \text{by definition of } p_x \\
&= L_{\max}^S + \Delta
\end{aligned}$$

Thus, we have shown that algorithm  $A$ , which is not a CL algorithm, does not produce an optimal schedule for  $I'$ .  $\square$

Theorem 4.2 shows that, to be optimal, an algorithm cannot deviate from CL rule at any time, even though the current critical compound laxity is large and the maximum lateness among jobs that have been completed so far is large. An interesting point to note is that if an online algorithm  $A$  deviates from the CL rule, the adversary can construct a counterexample by modifying the input instance in an *online* fashion; when algorithm  $A$  deviates from the CL rule, the adversary can generate one additional job to cause  $A$  to have a larger lateness while an optimal algorithm can still maintain a smaller lateness. Figure 2 illustrates the construction of a counterexample. Suppose an algorithm  $A$  does not follow the CL rule in some time interval while processing input instance  $I$ .  $A4$  shows the partial schedule for  $I$  up to time 4 and the compound laxity of available jobs. Job 1 has the smallest compound laxity, and the critical deadline is  $d_1 = 10$ . Violating the CL rule, algorithm  $A$  runs job 4 in the interval  $[4, 5)$ .  $A5$  shows the partial schedule for  $I$  up to time 5 and the compound laxity of available jobs. The critical compound laxity decreased from 4 to 3 between time 4 and 5. The adversary modifies the input instance  $I$  into a counterexample by first removing jobs that have not arrived (job 5) resulting in  $I_t$ , and then adding job 6 resulting in  $I'$ .  $A5'$  shows the partial schedule and the compound laxity of available jobs in instance  $I'$ . Note that jobs 3 and 6 have the same deadline. Thus, they have the same compound laxity of 1. The partial schedules for  $I$ ,  $I_t$ , and  $I'$  are the same up to time 5 because  $A$  cannot distinguish them until time 5. The partial schedule is shown in both  $A5$  and  $A5'$ .  $A(I')$  shows a schedule that  $A$  might construct for  $I'$ . Job 6 has a lateness of -1 yielding the maximum lateness.  $OPT(I')$  shows a CL schedule for  $I'$ . Jobs 1 and 6 have a lateness of -2 yielding the maximum lateness. It is left to the reader to verify that had the instance  $I$  not been modified into  $I'$ , algorithm  $A$  may be optimal.

## 5 Open Problems

In this section, some open problems are given. The first open problem is to somehow extend the results to multiple machines. Consider the problem of preemptively scheduling jobs that arrive over time on identical parallel machines so that all jobs finish by their deadlines. An online algorithm is

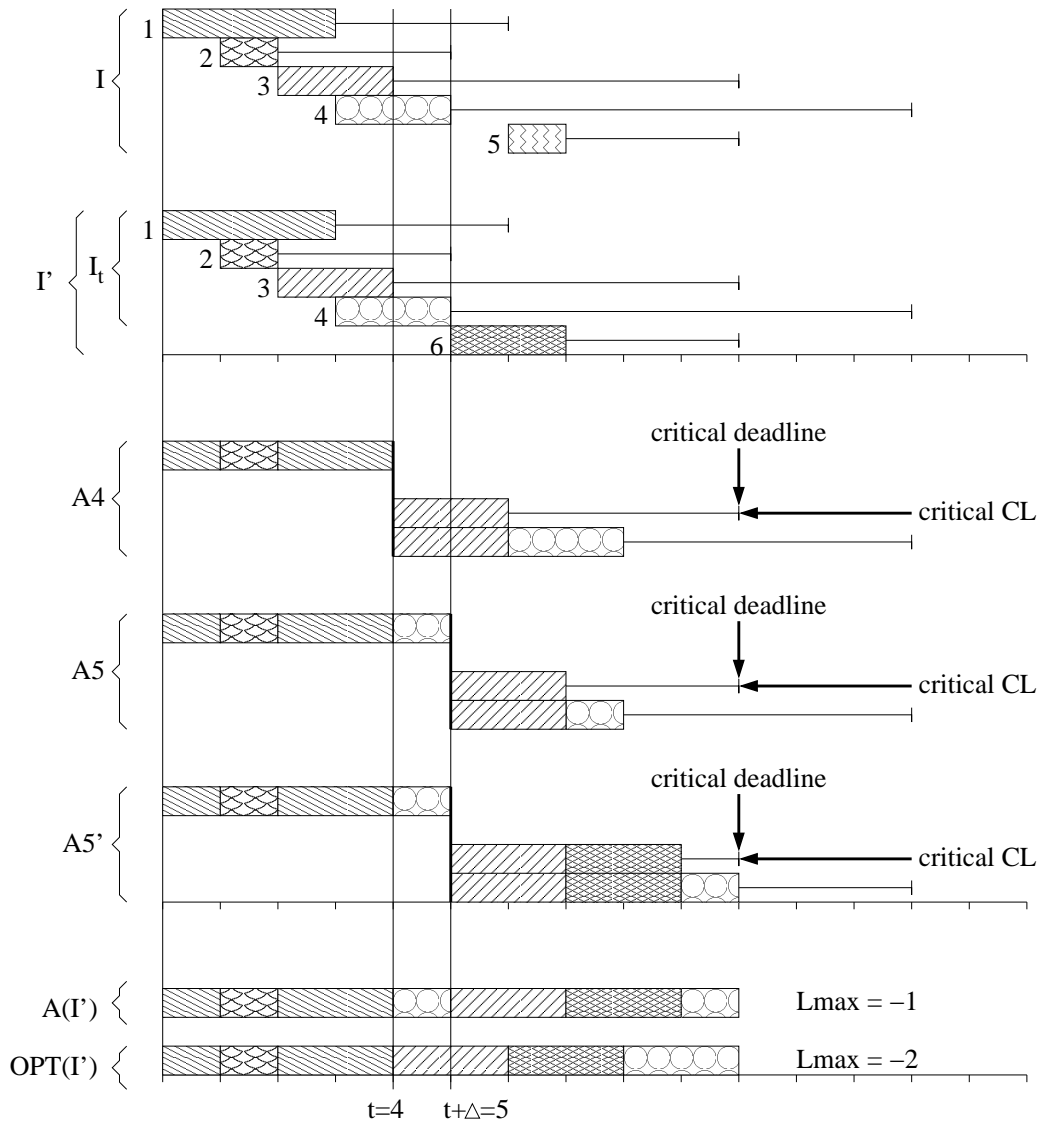


Figure 2: Example illustrating non-optimality of non-CL algorithms.

“admissible” if it can produce a feasible schedule whenever the optimal algorithm can. It is known that no online algorithm is admissible if it has the same number of machines as the optimal offline algorithm [8]. An open question is whether there is an online admissible algorithm that uses  $cm$  machines for some constant  $c$  while the optimal algorithm uses only  $m$  machines. Another way to augment an online algorithm is to use faster machines. Similar problems are considered in [7, 6, 2].

Another direction is to study problems of optimizing a second objective function subject to the constraint that the maximum lateness is minimum. With all the optimal online algorithms for the maximum lateness criterion identified, finding an algorithm to optimize the second objective will be more approachable. An example of work along this line is [9].

## References

- [1] K.R. Baker. *Introduction to Sequencing and Scheduling*, chapter 2. Wiley, New York, 1974.
- [2] Ho-Leung Chan, Tak-Wah Lam, and Kar-Keung To. Nonmigratory online deadline scheduling on multiprocessors. *Siam J. Comput.*, 34(3):669–682, 2005.
- [3] M. Dertouzos and A. Mok. Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Transactions on Software Engineering*, 15:1497–1506, 1989.
- [4] M. L. Dertouzos. Control robotics: the procedural control of physical processes. In *Proceeding of IFIP Congress*, pages 807–813, 1974.
- [5] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math*, 5:287–326, 1979.
- [6] Chiu-Yuen Koo, Tak-Wah Lam, Tsuen-Wan Ngan, and Kar-Keung To. Extra processors versus future information in optimal deadline scheduling. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures (SPAA)*, pages 133–142, 2002.
- [7] Tak Wah Lam and Kar-Keung To. Performance guarantee for online deadline scheduling in the presence of overload. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms*, pages 755–764, 2001.
- [8] Cynthia A. Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 140–149, 1997.
- [9] April Rasala, Cliff Stein, Eric Torng, and Patchrawat Uthaisombut. Existence theorems, lower bounds and algorithms for scheduling to meet two objectives. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 723–731, 2002.