

The Optimal Online Algorithms for Minimizing Maximum Lateness

Patchrawat Uthaisombut (utp@cs.pitt.edu)
Department of Computer Science, University of Pittsburgh

July 2, 2003

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. It was not previously known what other online algorithms are optimal for this problem. A complete characterization of all optimal online algorithms for this problem is given.

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 $1 | r_j, \text{pmtn} | L_{\max}$ according to Graham et. al.'s notation [GLLR79]. It is well known that the Earliest-Deadline-First (EDF) algorithm and the Least-Laxity-First (LLF) algorithm are optimal for this problem [Der74, DM89]. 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 $1 | r_j, \text{pmtn} | \sum_j C_j$ according to Graham et. al.'s notation [GLLR79]. The Smallest-Remaining-Processing-Time (SRPT) algorithm is optimal for this problem [Bak74]. 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 it 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 believably 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.

This rule has been discovered and is presented in this paper. It turns out that the general idea behind LLF is right. However, to be absolutely right, the remaining processing time of other jobs must be taken into account. This leads to the definition of the "compound laxity" of a job. This, in turn, leads to the "compound laxity rule" (CL rule) for choosing a job to run at any time. Interestingly, this rule has a flavor of EDF. The result is a complete characterization of all optimal online algorithms for the maximum lateness problem; for an online algorithm to be optimal, it must always follow the CL rule. To the best of the author's knowledge, this is the first non-trivial result of the sort.

Beside a better understanding of a fundamental problem and the discovery of its underlying optimal rule, this work also has potential leading up to future research. One 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 [RSTU02].

This work could be a basic building block for some other more complex problems. One problem is 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 "admissible" if it can produce a feasible schedule whenever the optimal algorithm can. An open question is whether there is an online admissible algorithm that uses a constant multiple of m machines while the optimal algorithm uses only m machines. [PSTW97]

The rest of the paper is organized as follows. In section 2, the definition of the problem and other quantities are given. The compound laxity and the compound laxity rule are defined. Section 3 furnishes the relationship between compound laxity and lateness. Main results are in section 4.

2. Definitions

The problem of preemptively scheduling jobs that arrive over time on a single machine to minimize maximum lateness is considered. This problem is denoted $1 | r_j, \text{pmtn} | L_{\max}$ according to Graham et. al.'s notation [GLLR79]. 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 $L_{\max}^S = \max_j L_j^S$. 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 .

For any input instance I , any schedule S for I , and any time t where $r_j \leq t \leq C_j^S$, let $p_j^S(t)$ be the remaining processing time of job j at time t in schedule S , and let $q_j^S(t)$ be the amount of work done on job j by time t in schedule S . Note that $p_j^S(t) + q_j^S(t) = p_j$ for any schedule S and any time t where $r_j \leq t \leq C_j^S$. For any non-empty subset X of jobs in I , let $r(X) = \min_{i \in X} r_i$, $p(X) = \sum_{i \in X} p_i$, and $d(X) = \max_{i \in X} d_i$. Also, let $p^A(X, t) = \sum_{i \in X} p_i^A(t)$ and $q^A(X, t) = \sum_{i \in X} q_i^A(t)$. Let $q_i^S(s, t) = q_i^S(t) - q_i^S(s)$. This is the amount of time the machine spends on job i during the interval $[s, t)$. Let $q^S(X, s, t) = \sum_{i \in X} q_i^S(s, t)$. This is the amount of time the machine spends on jobs in set X during the interval $[s, t)$.

Let $B_j(t)$ be the set of jobs i such that $r_i \leq t$ and $d_i \leq d_j$. Define the *compound laxity* of a job j at time t in schedule S , denoted $l_j^S(t)$, as $l_j^S(t) = d_j - t - p^S(B_j(t), t)$. Note that $d_i = d_j$ implies that $B_i(t) = B_j(t)$ and $l_i^S(t) = l_j^S(t)$. Define the *critical compound laxity* at time t in schedule S , denoted $l_{\min}^S(t)$, as $l_{\min}^S(t) = \min_j l_j^S(t)$ where the minimum is taken over the set of jobs that have arrived but have not completed in schedule S . Let $I_{\min}^S(t)$ be the set of all jobs i such that $l_i^S(t) = l_{\min}^S(t)$. Define the *critical deadline* at time t in schedule S , denoted $d_{\text{crit}}^S(t)$, as $d_{\text{crit}}^S(t) = \min_{i \in I_{\min}^S(t)} d_i$. For any times s and t such that $s \leq t$, let $B_j(s, t) = B_j(t) - B_j(s)$. This is the set of jobs i such that $s < r_i \leq t$ and $d_i \leq d_j$. For any jobs j and k such that $d_j \leq d_k$, let $B_{j,k}(t) = B_k(t) - B_j(t)$. This is the set of jobs i such that $r_i \leq t$ and $d_j < d_i \leq d_k$. By convention, if job j runs continuously from time s to time t , we will say that j runs in the close-open interval $[s, t)$.

A job is *available* at time t if it has arrived but not completed. An online algorithm A is a *CL algorithm* if for any input instance I and at any time t , algorithm A runs an available job j such that $d_j \leq d_{\text{crit}}^{A(I)}(t)$. We also call this the *CL rule*. A schedule S for an input instance I is a *CL schedule* if it can be produced using the CL rule.

At any time t , the Earliest-Deadline-First (EDF) algorithm runs an available job j such that

$d_j = \min_i d_i$ where the minimum is taken over all available jobs [Der74]. At any time t , the Least-Laxity-First (LLF) algorithm runs an available job j such that $d_j - t - p_j^{\text{LLF}}(t) = \min_i d_i - t - p_i^{\text{LLF}}(t)$ where the minimum is taken over all available jobs [DM89]. Both EDF and LLF are optimal for this problem. In the following, we will 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 1, which applies only to optimal schedules, all results in this section apply to any schedule. Lemma 2 establishes a relationship between the compound laxity of two different jobs. Lemma 3 tells us how the compound laxity of a job changes over time. An implication of Lemma 3 is that the compound laxity of a job never increases over time. This leads to Corollary 4, which states that the compound laxity of a job is smallest when it completes. Lemma 5 establishes a relationship between the compound laxity of a job and its lateness. Lemma 6 states that the maximum lateness equals the negation of the minimum compound laxity.

Lemma 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 starts 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, $r(X) + p(X) - d(X) \leq C_j^S - d_j = L_j^S \leq L_{\max}^S = L_{\max}^*(I)$. ■

Lemma 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: $l_j(t) - l_i(t) = [d_j - t - p(B_j(t), t)] - [d_i - t - p(B_i(t), t)] = d_j - d_i - [p(B_j(t), t) - p(B_i(t), t)] = d_j - d_i - p(B_{i,j}(t), t)$ ■

Lemma 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 $l_j^S(t) = l_j^S(s) - p(B_j(s, t)) - (t - s - q^S(B_j(t), s, t))$.

Proof:

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

$$\begin{aligned}
&= (t - s) - q(B_j(s), s, t) + p(B_j(s, t), t) \text{ by definition of } q(X, s, t) \\
&= (t - s) - q(B_j(s), s, t) + p(B_j(s, t)) - q(B_j(s, t), t) \text{ by definition of } p(X, t) \text{ and } q(X, t) \\
&= (t - s) - q(B_j(t), s, t) + p(B_j(s, t)) \text{ because } q(B_j(s, t), t) = q(B_j(s, t), s, t) \quad \blacksquare
\end{aligned}$$

Two factors control the change of the compound laxity of a job j over time: (1) arrival of new jobs i with $d_i \leq d_j$ and (2) the amount of time spent (or not spent) on available jobs with $d_i \leq d_j$. Lemma 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 sum of the total processing time of new jobs that arrive during the interval $[s, t)$ and the amount of time during $[s, t)$ that is NOT spent on jobs in $B_j(t)$.

Corollary 4: 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**, 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. \blacksquare

Lemma 5: 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 $L_j^S < -l_j^S(C_j^S)$, 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, $L_x = C_x - d_x \geq C_j + p(B_j(C_j), C_j) - d_j + (d_j - d_x) = -l_j(C_j) + (d_j - d_x) \geq -l_j^S(C_j^S)$. \blacksquare

Lemma 6: 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 will 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 5.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 5.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 5.b**, job k is the last job in $B_k(C_k)$ to finish.

To prove the other direction, assume that k is the last job to finish in $B_k(C_k)$ and $l_k(C_k) = \min_i l_i(C_i)$. Assume to reach a contradiction that $L_k < L_{\max}$. Then $L_k < L_{\max} = -\min_i l_i(C_i) = -l_k(C_k)$. From **Lemma 5.c**, there exists a job x such that $L_x \geq -l_k(C_k) > L_{\max}$. A contradiction. Thus, $L_k = L_{\max}$. ■

4. Results on Compound Laxity Algorithms

In this section, results on CL algorithms are given. Lemma 7 is the main lemma, which establishes a relationship similar to that in Lemma 1. Lemma 1 and Lemma 7 implies Theorem 8, which says that CL algorithms are optimal. In Lemma 9, EDF and LLF are shown to be CL algorithms. This implies that EDF and LLF are optimal as stated in Corollary 10. Corollary 11 states a basic property of EDF, which is used in Theorem 12 in showing that non-CL algorithms are not optimal.

Lemma 7: 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

$$d_u \leq d_k \tag{1}$$

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 Δ 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**, 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,

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

where the inequality follows from **Lemma 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 6**, $L_k = -l_k(C_k)$, and from **Lemma 5.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 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 4**, and the equality follows from **Lemma 6**. 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. ■

Theorem 8: 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 $1 \mid r_j, \text{pmtn} \mid L_{\max}$.

Proof: This follows from **Lemma 1**, **Lemma 7**, and the fact that $L_{\max}^*(I) \leq L_{\max}^{A(I)}$. ■

Lemma 9: 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. 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 $d_k < d_j$.

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

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

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

Corollary 10: EDF and LLF are optimal algorithms for the problem $1 \mid r_j, \text{pmtn} \mid L_{\max}$.

Proof: The results follow from **Lemma 9** and **Theorem 8**. \blacksquare

Corollary 11: 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 .

$$\begin{aligned}
l_j(t) &= l_j(s) - p(B_j(s, t)) - (t - s - q(B_j(t), s, t)) && \text{from Lemma 3} \\
&= l_j(s) - (t - s - q(B_j(t), s, t)) && \text{because no jobs arrive during } (s, t] \\
&= l_j(s) - (t - s - (t - s)) && \text{because EDF always runs a job} \\
&\quad \text{with the smallest deadline (no larger than } d_j), \text{ which must be in } B_j(t) \\
&= l_j(s) \quad \blacksquare
\end{aligned}$$

Theorem 12: If an online algorithm A is not a CL algorithm, it is not optimal for the problem $1 \mid r_j, \text{pmtn} \mid L_{\max}$.

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 never runs jobs in $B_w(t)$ 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 Δ 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 $r_x = t + \Delta$, $p_x = \max\{0, l_w^{A(I)}(t) + L_{\max}^*(I_t)\}$, and $d_x = d_w$. 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 A follows the CL rule during $[0, t)$, 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 during the interval $[0, t)$. Let X be the set of jobs that have finished by time $t+\Delta$ in schedule S . Let Y be the set of jobs that have not finished by time $t+\Delta$ in schedule S . For any job j in Y ,

$$\begin{aligned}
L_j^S &\leq -l_j^S(C_j^S) && \text{from Lemma 5. a} \\
&= -l_j^S(t + \Delta) && \text{from Corollary 11 and that no jobs arrive after time } t + \Delta \\
&= \begin{cases} -(l_j^S(t) - p_x) & \text{if } d_x \leq d_j \\ -l_j^S(t) & \text{if } d_j < d_x \end{cases} && \text{from Lemma 3, that EDF is used, job } x \text{ arrives at time } t + \Delta, \\
&\leq p_x - l_w^S(t) && \text{and no other jobs arrive during } (t, t + \Delta] \\
&= \max\{L_{\max}^*(I_t), -l_w^{A(I)}(t)\} && \text{because } l_w^S(t) = l_{\min}^S(t) \leq l_j^S(t) \\
& && \text{by definition of } p_x
\end{aligned}$$

Note that $L_i^S \leq L_{\max}^*(I_t)$ for all $i \in X$ because X is a subset of I_t and S is an optimal schedule. Thus, $L_{\max}^S = \max\{\max_{i \in X} L_i^S, \max_{i \in Y} L_i^S\} = \max\{L_{\max}^*(I_t), -l_w^{A(I)}(t)\}$.

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

$$\begin{aligned}
L_{\max}^{A(I')} &= -\min_j l_j^{A(I')}(C_j^{A(I')}) && \text{from Lemma 6} \\
&\geq -l_w^{A(I')}(t + \Delta) && \text{from Corollary 4} \\
&= -(l_w^{A(I')}(t) - p_x - \Delta) && \text{from Lemma 3, arrival of job } x, \text{ and that} \\
& && \text{jobs in } B_w(t + \Delta) = B_w(t) \text{ do not run during } [t, t + \Delta) \\
&= \max\{L_{\max}^*(I_t), -l_w^{A(I)}(t)\} + \Delta && \text{by definition of } p_x
\end{aligned}$$

Thus, we have shown that algorithm A does not produce an optimal schedule for I' . ■

Theorem 12 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 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 proof for the non-optimality of algorithm A in an online fashion. As algorithm A executes over time, as soon as an 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.

5. References

[Bak74] K.R. Baker. *Introduction to Sequencing and Scheduling*, chapter 2. Wiley, New York, 1974.

[Der74] M. Dertouzos. Control robotics: the procedural control of physical processes. In *Proc. IFIP Congress*, pages 807-813, 1974.

[DM89] M. Dertouzos and A. Mok. Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Transactions on Software Engineering*, 15:1497-1506, 1989.

[GLLR79] 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. *Annals of Discrete Mathematics*, 5:287-326, 1979.

[PSTW97] 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.

[RSTU02] 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.