# On the Intractability of Preemptive Single-Machine Job Scheduling with Release Times, Deadlines, and Family Setup Times

Abhishek Singh[a]

[a] *Washington University in Saint Louis, One Brookings Drive, Campus Box 1045, MO 63130, Saint Louis, USA*

**Abstract**

When building secure safety-critical software systems upon modern processors, a system designer may wish to thwart cache side-channel attacks while ensuring that no process misses its deadline. If the side-channel attacks are prevented in a uniprocessor environment by flushing the cache whenever context is switched from a process with a high security level to a process with a low security level then the problem of scheduling processes while meeting deadlines can be modeled as a single-machine job scheduling problem with release times, deadlines, preemption, and family setup times, which is known to be NP-hard. Since we expect the number of security levels and the worst-case cache flush time in practical applications to be "small", the number of families, $F$, and the largest setup time, $S$, are natural parameters for the problem. We show that the problem, when parameterized by $(F, S)$, is not fixed-parameter tractable by proving that the single-machine job scheduling problem with release times, deadlines, preemption, and two families with setup times picked from the set $\{0, 1\}$ is NP-hard. We also develop an $\mathcal{O}(n \log n)$ algorithm for the single-machine job scheduling problem with deadlines and two families with setup times.

*Keywords:* job scheduling, complexity, algorithm design, setup times, software security, side-channel attacks

*Email address:* abhishek.s@wustl.edu (Abhishek Singh)

## 1. Introduction

Software side-channel attacks can be constructed by leaking information between processes through the state of the processor's memory cache; in uniprocessor environments, an expensive, yet effective, countermeasure is to flush the cache on every context switch from one process to another [1]. In safety-critical software systems it is often the case that the processes must meet certain deadlines, and frequent cache flushes make it harder to achieve that objective. The frequency of cache flushes may be reduced by assigning a security level to each process and by flushing the cache only when context is switched from a process with a higher security level to a process with a lower security level [2].

In a job scheduling problem with setup times, a collection of jobs is partitioned into families such that changing over from one job to another takes negligible (zero) time within the same family but a setup time $s_{fg}$ is required when changing over from a job in family $f$ to a job in family $g$. If $s_{fg}$ is independent of $f$, then the setup times are said to be sequence-independent, and we may denote the setup time as simply $s_g$; otherwise, they are said to be sequence-dependent. The problem of scheduling processes in a uniprocessor environment with release times, deadlines, cache flush times, and security levels may be viewed as a special case of the single-machine job scheduling problem with release times, deadlines and family setup times:

- the single processor is mapped to a single machine;
- each process $p$ is mapped to a job $j_p$ (process characteristics like release times, deadlines, preemptivity are mapped to their equivalent job characteristics);
- each security level $l$ is mapped to a unique family $f_l$ such that if a process $p$ has security level $l$ then the corresponding job $j_p$ belongs to family $f_l$;
- for families $f$ and $g$, $s_{fg}$ is zero if $f$ does not have a higher security level than $g$; otherwise $s_{fg}$ equals $\delta$, where $\delta$ is the worst-case cache flush time for the architecture. In general, the setup times are sequence-dependent.

Crucially, observe that the number of families and the largest setup time in this mapping are expected to be "small". For example, the reference multilevel security policy in SELinux contains only 16 security levels, if we ignore category sets [3], and Xu et al. [4] observe that the `wbinvd` instruction (which writes back all the modified data in the cache and invalidates the entire shared cache on an x86 system) takes at most 0.7 ms for the platform used in their experiments. Thus, the number of families, denoted $F$, and the largest setup time, denoted $S$[1], are natural parameters of the problem. In this research, we wish to investigate the complexity of single-machine job scheduling problems with respect to these parameters.

### *1.1. Notation*

A scheduling problem is described as a triple $(\alpha \mid \beta \mid \gamma)$, where $\alpha$ denotes the machine environment, $\beta$ denotes the job characteristics, and $\gamma$ denotes the optimality criteria [5]. For all problems considered in this paper, $\alpha$ and $\gamma$ are fixed to 1 and $-$, i.e., we are only concerned with feasibility on a single machine with various combinations of the following job characteristics:

| Symbol | Job Characteristic |
|---|---|
| $r_j$ | release dates |
| $d_j$ | deadlines |
| pmtn | preemption is permitted |
| $s_f$ | sequence-independent family setup times |
| $s_{fg}$ | sequence-dependent family setup times |
| $F = \text{const}$ | constant number of families |
| $F = 2$ | two families |

### *1.2. Previous Work*

Ghosh and Gupta [6] developed an $\mathcal{O}(F^2 n^F)$ algorithm for $(1 \mid d_j, s_{fg} \mid L_{\max})$, which may be adapted to solve $(1 \mid d_j, s_{fg} \mid -)$ by comparing the output

[1] We assume that all input data including setup times is integral: thus, a problem instance cannot be scaled to make $S$ arbitrarily small, and $S$ is a meaningful parameter.

Table 1: Complexity classes assuming $F$ to be constant are shown. $A_1$ refers to the fact that in the absence of release times the existence of a preemptive schedule implies the existence of a non-preemptive schedule. $A_2$ refers to the fact that sequence-independent setup times are a special case of sequence-dependent setup times.

| Job Characteristics | | | | Complexity Class | Reason |
|---|---|---|---|---|---|
| $d_j$ | – | $s_f$ | – | P | By Row 3, $A_2$ |
| $d_j$ | – | $s_f$ | pmtn | P | By Row 1, $A_1$ |
| $d_j$ | – | $s_{fg}$ | – | P | From [6] |
| $d_j$ | – | $s_{fg}$ | pmtn | P | By Row 3, $A_1$ |
| $d_j$ | $r_j$ | $s_f$ | – | Strongly NP-hard | From [8] |
| $d_j$ | $r_j$ | $s_f$ | pmtn | Unknown | – |
| $d_j$ | $r_j$ | $s_{fg}$ | – | Strongly NP-hard | From [8] |
| $d_j$ | $r_j$ | $s_{fg}$ | pmtn | Unknown | – |

with zero. The running time of the algorithm indicates that instances of the problem with "very small" values of $F$ are tractable.

The main intractability result for job scheduling problems with setup times is due to Bruno and Downey [7] who showed that $(1 \mid d_j, s_f \mid -)$, is NP-hard. Since the reduction uses an arbitrary number of families, it does not shed any light on the tractability of $(1 \mid d_j, s_f, F = \text{const} \mid -)$.

If $F$ is assumed to be constant, then the only complexity result available to us is due to Garey and Johnson [8] who show that $(1 \mid r_j, d_j \mid -)$ is strongly NP-hard, which implies, for instance, that $(1 \mid r_j, d_j, s_f, F = \text{const} \mid -)$ is NP-hard. The complexity classes for all relevant combinations of job characteristics are listed in Table 1. Note that resolving the complexity class of $(1 \mid r_j, d_j, \text{pmtn}, s_f, F = \text{const} \mid -)$ is an open problem. The tractability of $(1 \mid r_j, d_j, \text{pmtn} \mid -)$[2] suggests that $(1 \mid r_j, d_j, \text{pmtn}, s_f, F = \text{const} \mid -)$ might be tractable. Our results show that this is not the case.

### *1.3. Our Results*

Our first result is that $(1 \mid r_j, d_j, \text{pmtn}, s_f \mid -)$ is weakly NP-hard even in the following restricted case:

[2]The fundamental preemptive scheduling problem admits a polynomial-time solution that greedily uses the ED (earliest deadline) order; other techniques like linear programming and bipartite matching may also be used.

- there are two families $h$ and $l$; and
- one family $h$ has no setup time while the other family $l$ has unit setup time, i.e., $s_h = 0$ and $s_l = 1$.

Since sequence-independent setup times are a special case of sequence-dependent setup times, $(1 \mid r_j, d_j, \text{pmtn}, s_{fg} \mid -)$ is also NP-hard in the same restricted case. Therefore, the two unclassified problems in Table 1 are resolved to be weakly NP-hard.

To gain further appreciation for the importance of the above result, some readers may find it useful to view it through the lens of parameterized complexity. A problem with parameter $k$ is considered to be *fixed-parameter tractable* if it can be solved in $O(f(k) \cdot |x|^c)$ time, where $x$ is an input, $|x|$ is its size, $f$ is some computable function, and $c$ is some constant. All fixed-parameter tractable problems lie in the (parameterized complexity) class FPT, which is an analogue of the classical complexity class P. NP-hard problems like VERTEX COVER are known to be in FPT. Intractability of parameterized problems is captured by a collection of complexity classes called the W-hierarchy which is contained in the large class XP, an analogue of the classical complexity class EXP:

$$\mathsf{FPT} \subseteq \mathsf{W}[1] \subseteq \mathsf{W}[2] \subseteq \cdots \subseteq \mathsf{XP}.$$

Problems like CLIQUE and DOMINATING SET occupy the lower levels of the W-hierarchy, and are believed to be parametrically intractable. The class XP contains any problem that can be solved by an algorithm in $O(f(k) \cdot |x|^{g(k)})$ time, where $x$ and $k$ are the input and parameter respectively, and $f$ and $g$ are some computable functions. The NP-hardness result in the previous paragraph implies that such an algorithm cannot exist for $(1 \mid r_j, d_j, \text{pmtn}, s_f \mid -)$ when the parameter is chosen to be $(F, S)$, unless $\mathsf{P} = \mathsf{NP}$, because $F = 2$ and $S = 1$ in the restricted case considered in the result. Thus, assuming $\mathsf{P} \neq \mathsf{NP}$, $(1 \mid r_j, d_j, \text{pmtn}, s_f \mid -)$, parameterized by $(F, S)$, is not in XP, and is highly intractable. While the results in this paper can be understood without the aid of parameterized complexity, the discussion in this paragraph provides a different

and interesting vantage point for viewing the same results. Our introduction to parameterized complexity classes has been brief, and the interested reader can find more information in standard texts [9, 10].

Our second result is an $\mathcal{O}(n \log n)$ time algorithm for $(1 \mid d_j, \text{pmtn}, s_f, F = 2 \mid -)$, which is a significant improvement over the $\mathcal{O}(n^2)$ algorithm of Ghosh and Gupta [6].

## 2. The Intractability Result

To prove NP-hardness of the problem $(1 \mid r_j, d_j, \text{pmtn}, s_f, F = 2 \mid -)$, we show that PARTITION [11], a weakly NP-hard problem, is polynomial-time reducible to the problem. We consider the following version of PARTITION:

- *Instance:* A set of positive integers $A = \{a_1, a_2, \ldots, a_m\}$.
- *Question:* Does there exist a subset $A' \subseteq A$ such that

$$\sum_{a \in A'} a = \sum_{a \in A \setminus A'} a,$$

and $|A'| = m/2$?

We transform an instance of PARTITION to an instance of our problem in which there are two families $h$ and $l$ such that $s_h = 0$ and $s_l = 1$. Each $a_i$ is mapped to an interval $I_i$ of length $X_i = a_i + 3M + 7$, where $M = 1 + \sum_{i=1}^{m} a_i$. The intervals are placed next to each other in the order $I_1, I_2, \ldots, I_m$; thus, the right end of the interval $I_i$ equals the left end of the interval $I_{i+1}$ for all $i \in \{1, 2, \ldots, m-1\}$. Any interval $I_i$ contains four fixed unit jobs $u_1$, $u_2$, $u_3$, and $u_4$:

| Job | $r$ | $p$ | $d$ | $f$ |
|---|---|---|---|---|
| $u_1$ | $0$ | $1$ | $1$ | $h$ |
| $u_2$ | $M + 2$ | $1$ | $M + 3$ | $h$ |
| $u_3$ | $X_i - M - 3$ | $1$ | $X_i - M - 2$ | $l$ |
| $u_4$ | $X_i - 1$ | $1$ | $X_i$ | $l$ |

Thus, $u_1$ is a job in family $h$ with release time 0, deadline 1, and unit execution time, and the other records may be read similarly. The release times and deadlines are relative to the interval $I_i$: thus, if $I_i$ begins at time $t$, then $u_3$ has a release time at $t + X_i - M - 3$. Since the first job of any interval is in family $h$ and $s_h = 0$, we can safely place the intervals right next to each other. $I_i$ contains two more jobs, $v_1$ and $v_2$:

| Job | $r$ | $p$ | $d$ | $f$ |
|---|---|---|---|---|
| $v_1$ | 2 | $M$ | $2M + 4$ | $l$ |
| $v_2$ | $X_i - 2M - 4$ | $M$ | $X_i - 2$ | $h$ |

Thus, $v_1$ is a job in family $l$ with release time 2, deadline $2M+4$, and processing time $M$, and the other record may be read similarly. The internal structure of $I_i$ is determined by the four fixed unit jobs, $u_1$, $u_2$, $u_3$, and $u_4$, and the two jobs, $v_1$ and $v_2$ with larger spans. The following lemma may be verified by performing some elementary algebra.

**Lemma 1.** *The following equations hold for the jobs $u_1$, $u_2$, $u_3$, $u_4$, $v_1$, and $v_2$:*

$$\begin{aligned} r(u_2) - d(u_1) &= M + 1 \\ r(v_2) - d(u_2) &= a_i \\ r(u_3) - d(u_2) &= a_i + M + 1 \\ r(u_3) - d(v_1) &= a_i \\ r(u_4) - d(u_3) &= M + 1 \end{aligned}$$

Three configurations of the interval $I_i$ in which $v_1$ and $v_2$ are scheduled non-preemptively are shown in Figure 1. The configuration shown in the top row is labeled *lhhl* because the jobs $u_2$, $v_1$, $v_2$, $u_3$ scheduled in this order belong to the families $l, h, h, l$ respectively. While two more configurations, *hllh* and *lhlh*, are shown in the figure, the configuration *hlhl* is missing: visually, it is clear that the configuration is not valid since the interval $[d(u_2), r(u_3)]$ is not long enough to contain both $v_1$ and $v_2$. We can confirm this algebraically:

$$p(v_1) + p(v_2) = 2M > a_i + M + 1 = r(u_3) - d(u_2).$$

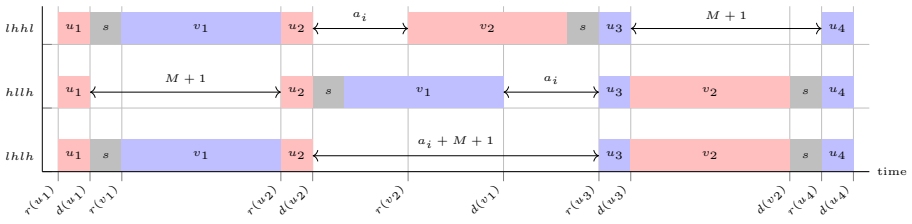


Figure 1: Three configurations of interval $I_i$ are shown here. The colors indicate the family of the jobs: red for $h$, blue for $l$, and grey for setup.

The first equality follows from the definitions of $v_1$ and $v_2$; the inequality follows from the definition of $M$; the second equality is taken from Lemma 1. Therefore, the three configurations that are shown in Figure 1 are the only viable configurations in which $v_1$ and $v_2$ are scheduled non-preemptively. Since preemption is permitted in the problem, many valid configurations are not shown in the figure: once we complete our construction, we will be able to show that the three configurations that are shown in Figure 1 are the only viable configurations for any interval $I_i$.

The final ingredients in our construction are two big jobs, $b_1$ and $b_2$ that span from the left end of the first interval $I_1$ to the right end of the last interval $I_m$:

| Job | $r$ | $p$ | $d$ | $f$ |
|---|---|---|---|---|
| $b_1$ | 0 | $\sum_{i=1}^{m}(a_i + M + 1)/2$ | $\sum_{i=1}^{m} X_i$ | $l$ |
| $b_2$ | 0 | $\sum_{i=1}^{m}(a_i + M + 1)/2$ | $\sum_{i=1}^{m} X_i$ | $h$ |

Note that unlike the definitions of the previous jobs, the release times and deadlines of $b_1$ and $b_2$ are in absolute coordinates. We need a few more auxiliary lemmas before we can prove our main result.

**Lemma 2.** *The total slack in our construction is* $2m$.

*Proof.* The total computational demand within each interval $I_i$ is

$$\sum_{i=1}^{4} p(u_i) + \sum_{j=1}^{2} p(v_j) = 4 + 2M.$$

The total computational demand from the big jobs, $b_1$ and $b_2$, is $\sum_{i=1}^{m}(a_i+M+1)$. Summing the two demands, we get a total demand of

$$\sum_{i=1}^{m}(a_i+3M+5)=\sum_{i=1}^{m}(X_i-2)=\sum_{i=1}^{m}(X_i)-2m. \qquad \square$$

**Lemma 3.** *Each interval $I_i$ contains at least two setups.*

*Proof.* In any $I_i$, there are three subintervals that lie between the four fixed jobs $u_1$, $u_2$, $u_3$, and $u_4$ (see Figure 1). There is at least one setup in the middle subinterval since $u_2$ and $u_3$ belong to families $h$ and $l$ respectively. Assume, for the sake of contradiction, that the left and right subintervals contain no setups. Then, $v_1$ and $v_2$ must be scheduled in the middle subinterval. This implies that $I_i$ is in the *hlhl* configuration, but we argued earlier that this configuration is invalid: thus, we have a contradiction. $\square$

**Lemma 4.** *Each interval must be in configuration lhhl or in configuration hllh.*

*Proof.* From Lemma 2 and Lemma 3, it follows that each interval $I_i$ contains exactly two setups. The arguments in the proof of Lemma 3 may be repeated to show that one setup must be in the middle subinterval, and the other setup must be in either the left subinterval or the right subinterval.

Assume that the left subinterval does not contain setups. Then, $v_1$ must be scheduled non-preemptively in the middle interval, separated from $u_2$ by a setup. Since the middle interval contains only one setup, there can be no setup between $d(v_1)$ and $r(u_3)$, which implies that $v_2$ must be scheduled non-preemptively in the right subinterval. Therefore, we end up in configuration *hllh*.

Symmetrically, assuming that the right subinterval does not contain setups, then we necessarily end up in configuration *lhhl*. $\square$

**Lemma 5.** *A schedule exists for the jobs if and only if the* PARTITION *instance $a_1, a_2, \ldots, a_m$ has a solution.*

*Proof.* From Lemma 4, we know that each interval $I_i$ is in configuration *lhhl* or in configuration *hllh*. To avoid introducing more setups in the configuration *lhhl*, $b_1$ must be scheduled between $d(u_3)$ and $r(u_4)$, and $b_2$ must be scheduled between $d(u_2)$ and $r(v_2)$. Thus, $b_1$ and $b_2$ execute for $M+1$ and $a_i$ units of time respectively in the configuration *lhhl*, using Lemma 1. Similarly, it can be shown that $b_1$ and $b_2$ execute for $a_i$ and $M+1$ units of time respectively in the configuration *hllh*. Now, we are ready to prove the two directions in the statement of the lemma.

( $\Longleftarrow$ ). For every $a_i \in A'$, set the configuration of $I_i$ to *lhhl*, and, for all remaining intervals, choose the other configuration. For each $a_i \in A'$, $b_1$ executes for $M+1$ units of time in $I_i$. Since $|A'| = m/2$, $b_1$ executes for a total of $(m/2)(M+1)$ units of time in these intervals. For each $a_i \notin A'$, $b_1$ executes for $a_i$ units of time, yielding a total execution time of

$$\sum_{a \notin A'} a_i = (\sum_{a \in A} a_i)/2.$$

The equality holds because $A'$ is a solution to the PARTITION instance. Over all intervals, $b_1$ executes for the following amount of time:

$$\begin{aligned} & (m/2)(M+1) + (\sum_{a \in A} a_i)/2 \\ = & \sum_{a \in A} (a_i + M + 1)/2 \\ = & \, p(b_1). \end{aligned}$$

Similarly, we can show that $b_2$ executes for $p(b_2)$ units of time. Thus, we have produced a feasible schedule.

($\implies$). Without loss of generality, we assume that intervals $I_1, I_2, \ldots, I_k$ are in configuration *lhhl* and the remaining intervals are in configuration *hllh*; this assumption can always be realized by renaming the intervals. Then, for each $i \in \{1, 2, \ldots, k\}$, $b_2$ executes for $a_i$ units of time in the interval $I_i$, and for each $i \in \{k+1, k+2, \ldots, m\}$, $b_2$ executes for $M+1$ units of time. Since this is a valid schedule, these execution times must sum to $p(b_2)$:

$$\sum_{i=1}^{k} a_i + \sum_{j=k+1}^{m} (M+1) = \sum_{i=1}^{m} (a_i + M + 1)/2 \tag{1}$$

$$\implies |(m/2 - k)(M+1)| = |\sum_{i=k+1}^{m} a_i/2 - \sum_{i=1}^{k} a_i/2|$$

$$\implies |(m/2 - k)(M+1)| < \sum_{i=1}^{m} a_i \qquad \text{(since } a_i \text{ are positive integers)}$$

$$\implies |(m/2 - k)(M+1)| < M \tag{2}$$

$$\implies k = m/2.$$

The last step uses the following reasoning: if $k \neq m/2$, then the lhs in Inequality 2 would be a positive multiple of $M+1$ and hence would exceed the rhs, yielding a contradiction. Setting $k = m/2$ in Equation 1, we get

$$\sum_{i=1}^{k} a_i = (\sum_{i=1}^{m} a_i)/2.$$

Therefore, $A' = \{a_1, a_2, \ldots, a_k\}$ is a valid solution for the PARTITION instance. □

**Theorem 6.** $(1 \mid r_j, d_j, \text{pmtn}, s_f \mid -)$ *is* NP*-hard even for the restricted case of two families and one unit setup time.*

*Proof.* The transformation of the PARTITION instance to the $(1 \mid r_j, d_j, \text{pmtn}, s_f \mid -)$ instance can be carried out in polynomial time, and from Lemma 5, it follows that the transformation is a valid reduction. □

## 3. The Algorithmic Result

In an instance of $(1 \mid d_j, \text{pmtn}, s_f, F = 2 \mid -)$, we have $n$ jobs partitioned into two families $h$ and $l$ with setup times $s_h$ and $s_l$ respectively. Before describing an $\mathcal{O}(n \log n)$ algorithm for $(1 \mid d_j, \text{pmtn}, s_f, F = 2 \mid -)$, we need a few lemmas.

**Lemma 7.** *There exists a schedule where the jobs within each family are scheduled according to the earliest deadline (ED)*[3] *order.*

*Proof.* This fact about scheduling with setup times can be proved using simple exchange arguments that preserve feasibility [12, Th. 1]. □

Let the jobs in each family be labeled according to the ED order: thus, for any $i, j \in h$, $d_i \leq d_j$ if $i < j$ (equal deadline ties are broken arbitrarily). Using Lemma 7, there exists a schedule such that $i$ precedes $j$ in the schedule, denoted $i \to j$, if $i < j$. For any $i, j$ such that $i \to j$, $d_i$ can be modified to reflect the precedence constraint [5, Ch. 3, p. 2]:

$$d_i \coloneqq \min\{d_i, d_j - p_j\}.$$

For each family, the digraph representing the precedence relations is a chain. The deadlines of all jobs may be updated systematically by updating the deadline of the last sink vertex on the chain, deleting the said vertex, and continuing the process. This takes $\mathcal{O}(n)$ time, and sorting the jobs in ED order takes $\mathcal{O}(n \log n)$ time: thus, the updates take $\mathcal{O}(n \log n)$ time.

Any schedule is made up of blocks where jobs of one family execute within a block. We refer to a block in which jobs from family $h$ ($l$, resp.) are scheduled as an $h$-block ($l$-block, resp.). If an $h$-block follows an $l$-block in a schedule, then there must be a gap of $s_h$ units between the two blocks; we refer to this portion of the schedule as an $lh$-transition. Similarly, an $hl$-transition denotes an $l$-block following an $h$-block with a gap of $s_l$ units in between.

**Lemma 8.** *There exists a schedule where at least one job completes in each block.*

[3] This is also called the earliest due date (EDD) order, or the earliest deadline first (EDF) order by various authors.

*Proof.* Consider an $h$-block in which some jobs are scheduled but none of them complete. Let us assume that jobs $j_1, j_2, \ldots, j_m$ are scheduled in the block from left to right, for some $m > 0$. Let the sub-block corresponding to $j_m$ extend over the interval $[t_1, t_2]$. Since $j_m$ does not complete in the block, it must resume execution at some time $t_3$ such that $t_3 > t_2$. We claim that the partial schedules over the intervals $[t_1, t_2]$ and $[t_2, t_3]$ may be safely exchanged with each other without violating setup constraints or affecting feasibility. By repeatedly performing such exchanges for $m-1, m-2, \ldots, 1$, we can eliminate the $h$-block from the schedule. □

**Lemma 9.** *There exists a schedule such that for any $fg$-transition where $f, g \in \{l, h\}$, one of the following conditions is true:*

*(i) all jobs in $f$ have been completed at the end of the $f$-block; or*

*(ii) the first job scheduled in the $g$-block completes at its deadline.*

*Proof.* Without loss of generality, we may restrict our attention to schedules where all jobs within a family are scheduled in ED order (using Lemma 7) and at least one job completes in each block (using Lemma 8). Consider the first $fg$-transition in a schedule where both conditions (i) and (ii) are false. Thus, at the end of the $f$-block there exists a job $i$ (with the earliest deadline) in $f$ that has not been completed, and the first job $j$ in the $g$-block completes before its deadline. Let $\varepsilon$ be defined as follows:

$$\varepsilon = \min\{d_j - C_j, p_i^*\},$$

where $C_j$ is the completion time of $j$ and $p_i^*$ is the remaining processing time of $i$. From the deadline modifications that we made earlier, it follows that we can shift the $g$-block $\varepsilon$ units to the right; we can schedule $\varepsilon$ units of $i$ in the space created by the shift. If one of the conditions in the statement of the lemma is true after this exchange, then we are done; otherwise, we can repeat the process. Since each iteration of the process increases the number of completed jobs in $f$, the repetitions cannot be carried out indefinitely. Therefore, one of the conditions will eventually be true. □

A simple scheduling algorithm follows from Lemma 9: keep scheduling jobs from family $f$ until time $d_j - p_j^* - s_g$ where $j$ is the job in family $g$ with the earliest deadline and with nonzero remaining processing time $p_j^*$. This consumes $\mathcal{O}(n)$ time. It is not clear whether the first block in the schedule should be an $h$-block or an $l$-block: we can run the algorithm for both choices. The modification of the deadlines takes $\mathcal{O}(n \log n)$ time, and it dominates the running time of the algorithm.

**Theorem 10.** *The algorithm described in this section solves* $(1 \mid d_j, \text{pmtn}, s_f, F = 2 \mid -)$ *in* $\mathcal{O}(n \log n)$ *time.*

## 4. Conclusion & Future Directions

We have shown that the problem $(1 \mid r_j, d_j, \text{pmtn}, s_f \mid -)$ is not fixed-parameter tractable when parameterized by $(F, S)$, unless $\mathsf{P} = \mathsf{NP}$. However, we still need to find acceptable solutions for practical applications. Questions in the vein of the following question need to be investigated:

> **Q1:** Are there other choices of parameters for the problem $(1 \mid r_j, d_j, \text{pmtn}, s_f \mid -)$ that lead to tractability, e.g., the number of distinct release times? Some such parameters have been investigated for hard scheduling problems [13].

Our algorithmic result shows that unlike $(1 \mid r_j, d_j, \text{pmtn}, s_f, F = 2 \mid -)$, the problem $(1 \mid d_j, \text{pmtn}, s_f, F = 2 \mid -)$ is tractable (in $\mathsf{P}$). Ghosh and Gupta's algorithm [6] shows that $(1 \mid d_j, \text{pmtn}, s_f \mid -)$ parameterized by $F$ is in $\mathsf{XP}$. These facts suggest the following open question:

> **Q2:** Is $(1 \mid d_j, \text{pmtn}, s_f \mid -)$ parameterized by $F$ in $\mathsf{FPT}$?

## 5. Acknowledgments

We thank the reviewers for their feedback which has helped in clearly presenting the results. This work was supported in part by the National Science Foundation (NSF), United States of America (grant number CPS-1911460).

## References

[1] D. A. Osvik, A. Shamir, E. Tromer, Cache Attacks and Countermeasures: The Case of AES, in: D. Pointcheval (Ed.), Topics in Cryptology – CT-RSA 2006, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2006, pp. 1–20. doi:10.1007/11605805_1.

[2] S. Mohan, M. K. Yoon, R. Pellizzoni, R. Bobba, Real-Time Systems Security through Scheduler Constraints, in: 2014 26th Euromicro Conference on Real-Time Systems, 2014, pp. 129–140. doi:10.1109/ECRTS.2014.28.

[3] B. Hicks, S. Rueda, L. St.Clair, T. Jaeger, P. McDaniel, A logical specification and analysis for SELinux MLS policy, ACM Transactions on Information and System Security 13 (3) (2010) 26:1–26:31. doi:10.1145/1805974.1805982.

[4] M. Xu, L. Thi, X. Phan, H.-Y. Choi, I. Lee, vCAT: Dynamic Cache Management Using CAT Virtualization, in: 2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2017, pp. 211–222. doi:10.1109/RTAS.2017.15.

[5] J. K. Lenstra, D. B. Shmoys, Elements of Scheduling, arXiv:2001.06005 [cs]arXiv:2001.06005.

[6] J. B. Ghosh, J. N. Gupta, Batch scheduling to minimize maximum lateness, Operations Research Letters 21 (2) (1997) 77–80. doi:10.1016/S0167-6377(97)00028-X.

[7] J. Bruno, P. Downey, Complexity of Task Sequencing with Deadlines, Set-Up Times and Changeover Costs, SIAM Journal on Computing 7 (4) (1978) 393–404. doi:10.1137/0207031.

[8] M. R. Garey, D. S. Johnson, Two-Processor Scheduling with Start-Times and Deadlines, SIAM Journal on Computing 6 (3) (1977) 416–426. doi:10.1137/0206029.

[9] R. G. Downey, M. R. Fellows, Parameterized Complexity, Springer Publishing Company, Incorporated, 2012.

[10] J. Flum, M. Grohe, Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series), Springer-Verlag, Berlin, Heidelberg, 2006.

[11] M. R. Garey, D. S. Johnson, Computers and Intractability; A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., USA, 1990.

[12] C. L. Monma, C. N. Potts, On the Complexity of Scheduling with Batch Setup Times, Operations Research 37 (5) (1989) 798–804. doi:10.1287/opre.37.5.798.

[13] M. Mnich, A. Wiese, Scheduling and fixed-parameter tractability, Mathematical Programming: Series A and B 154 (1-2) (2015) 533–562. doi:10.1007/s10107-014-0830-9.