


Scheduling on Identical Machines with Setup Time and Unknown Execution Time

Yasushi Kawase  

The University of Tokyo, Japan

Kazuhisa Makino 

Kyoto University, Japan

Vinh Long Phan 

Toyota Motor Corporation, Japan

Hanna Sumita 

Institute of Science Tokyo, Japan

Abstract

In this study, we investigate a scheduling problem on identical machines in which jobs require initial setup before execution. We assume that an algorithm can dynamically form a batch (i.e., a collection of jobs to be processed together) from the remaining jobs. The setup time is modeled as a known monotone function of the set of jobs within a batch, while the execution time of each job remains unknown until completion. This uncertainty poses significant challenges for minimizing the makespan. We address these challenges by considering two scenarios: each job batch must be assigned to a single machine, or a batch may be distributed across multiple machines. For both scenarios, we analyze settings with and without preemption. Across these four settings, we design online algorithms that achieve asymptotically optimal competitive ratios with respect to both the number of jobs and the number of machines.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online scheduling, Competitive analysis, Makespan minimization, Identical machines scheduling

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.41

Related Version *Full Version*: <https://arxiv.org/abs/2507.11311>

Funding This work was partially supported by the joint project of Kyoto University and Toyota Motor Corporation, titled “Advanced Mathematical Science for Mobility Society”, JST ERATO Grant Number JPMJER2301, JST PRESTO Grant Number JPMJPR2122, and JSPS KAKENHI Grant Numbers JP17K12646, JP19K22841, JP20H00609, JP20H05967, JP20K19739, JP21K17708, JP21H03397, and JP25K00137.

1 Introduction

Efficient job allocation across multiple workers or machines is crucial for optimizing productivity in industrial environments. For example, consider distributing computational jobs across multiple virtual machines. The processing time for a batch of jobs consists of two components: the *setup time*, which includes configuring the environment or installing necessary software or libraries, and the *execution time*, which represents the duration of performing the actual tasks. While setup times are typically known in advance, the execution time of each job is often unpredictable until the job is completed. This uncertainty complicates the design of scheduling algorithms aimed at minimizing the *makespan*, which is the time when all jobs are completed.

In this paper, we introduce a scheduling problem involving n jobs on m identical machines with known setup times and unknown execution times, referred to as the *unknown execution time scheduling (UETS)* problem. We assume that an algorithm can dynamically form a



© Yasushi Kawase, Kazuhisa Makino, Vinh Long Phan, and Hanna Sumita;
licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Algorithms and Data Structures (WADS 2025).

Editors: Pat Morin and Eunjin Oh; Article No. 41; pp. 41:1–41:16

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

batch – a collection of jobs that are processed together – from the remaining jobs. The setup time for a batch is defined as a set function over subsets of jobs, representing the total time required to prepare the jobs in the batch for processing. We explore two primary scenarios: one called *sUETS*, in which each constructed batch must be assigned to a single machine, and the other called *mUETS*, in which a batch can be distributed across multiple machines. Additionally, we examine two settings based on whether preemption is allowed. In the non-preemptive setting, once a batch process is started, it must be run without interruption until completion. In the preemptive setting, the processing of a batch can be interrupted. After an interruption, only uncompleted jobs are regrouped into batches and the process is resumed from the setup phase (completed jobs do not need to be processed again). Note that, the non-preemptive setting generally requires longer processing times compared to the preemptive one, because preemption allows for more flexible handling of incomplete batches and may reduce redundant processing.

Our objective is to design online algorithms for these scheduling problems. We analyze the performance of an online algorithm by the competitive ratio, which is the worst-case ratio between the makespan achieved by the online algorithm and that of an optimal offline algorithm. We assume that offline algorithms have complete knowledge of each job's execution time in advance. We refer to the schedule produced by the optimal offline algorithm as the *optimal schedule* and its makespan as the *optimal makespan*. We refer to the makespan of the schedule produced by an algorithm simply as the algorithm's makespan. An online algorithm is said to be ρ -competitive if its makespan is at most ρ times the optimal makespan for any instance. We will design online algorithms with asymptotically optimal competitive ratios with respect to both the number of machines and the number of jobs.

1.1 Our results

We study the competitive ratios for the scheduling problem with setup time and unknown execution times. We first show that any algorithm designed for jobs with a release time of 0 can be adapted to handle arbitrary release times at the expense of only a constant factor increase in the competitive ratio. We formally state this in Theorem 2. Thus, we may assume that the release time of each job is 0, i.e., jobs can start processing immediately at time 0. A summary of our results is provided in Table 1.

For the non-preemptive *sUETS* problem, we present two algorithms. First, we construct an algorithm with a competitive ratio of m (Theorem 3). Second, we design an algorithm that is $O(\sqrt{n/m})$ -competitive (Theorem 5). By combining these two algorithms, we obtain an $O(n^{1/3})$ -competitive algorithm (Corollary 6). Moreover, we prove lower bounds showing that every online algorithm for this problem must have a competitive ratio of at least $\Omega(m)$ (Theorem 11) and $\Omega(n^{1/3})$ (Theorem 10). For the preemptive *sUETS* problem, we design an algorithm with a competitive ratio of $O(\log n / \log \log n)$ (Theorem 8), and we demonstrate that the competitive ratios of $O(m)$ and $O(\log n / \log \log n)$ are best possible (Theorem 11).

Turning to the non-preemptive *mUETS* problem, we first construct an $O(\sqrt{m})$ -competitive algorithm (Theorem 12). By integrating this with the $O(\sqrt{n/m})$ -competitive algorithm for the non-preemptive *sUETS* problem (Theorem 5), we derive an $O(n^{1/4})$ -competitive algorithm for the non-preemptive *mUETS* problem (Corollary 13). We further prove that $O(\sqrt{m})$ and $O(n^{1/4})$ are optimal (Theorem 17). Finally, for the preemptive *mUETS* problem, we establish that the best possible competitive ratios are $\Theta(\log m / \log \log m)$ and $\Theta(\log n / \log \log n)$ (Theorems 8, 15, and 18).

Due to space limitations, some proofs are omitted in this version. The complete proofs are available in a full version of the paper.

■ **Table 1** The competitive ratios of the UETS problems with n jobs and m machines.

	single	multiple
non-preemptive	$\Theta(m)$ (Thms. 3 and 10) $\Theta(n^{1/3})$ (Cor. 6 and Thm. 10)	$\Theta(\sqrt{m})$ (Thms. 12 and 17) $\Theta(n^{1/4})$ (Cor. 13 and Thm. 17)
preemptive	$\Theta(m)$ (Thms. 3 and 11) $\Theta\left(\frac{\log n}{\log \log n}\right)$ (Thms. 8 and 11)	$\Theta\left(\frac{\log m}{\log \log m}\right)$ (Thms. 15 and 18) $\Theta\left(\frac{\log n}{\log \log n}\right)$ (Thms. 8 and 18)

1.2 Related work

Classical scheduling problems typically assume that the setup time is 0 for every job batch. In this case, the well-known *list scheduling algorithm* achieves $(2 - 1/m)$ -competitive, which is proven to be optimal [16–18, 32]. This algorithm assigns unprocessed jobs to any available machine in the order they appear on the job list, disregarding execution times.

When execution times are unknown until job completion, the problem falls under the category of *non-clairvoyant scheduling* [26]. Recent studies have explored non-clairvoyant scheduling in input prediction models [3, 21]. Shmoys et al. [32] introduced a technique to transform a ρ -competitive algorithm for a scheduling problem without release times into a 2ρ -competitive algorithm for the same problem with release times. For comprehensive overviews of online scheduling, see surveys [29, 31] and the book by Pinedo [28].

Scheduling problems that incorporate both setup times and execution times arise in various applications, such as cloud computing (where virtual machines must be initialized based on job types) and production systems (where machines require reconfiguration, such as changing molds or colors) [1, 9, 13, 20, 25, 30]. For example, in plastic production systems, attaching a specific mold to a machine constitutes the setup time. If consecutive jobs use the same mold, no additional setup time is required. The total setup time for a batch can be precomputed as the minimum time needed for attaching, exchanging, and removing molds.

Gambosi and Nicosia [13] studied an online version of scheduling with setup times in the one-by-one model. Mäcker et al. [24] investigated non-clairvoyant scheduling with setup times and proposed an $O(\sqrt{n})$ -competitive algorithm for minimizing maximum flow time on a single machine. Dogeas et al. [10] considered a scenario where the execution time of each job is only revealed after an obligatory test with a known duration. This test time can be viewed as a kind of setup time, although it differs in that the execution time becomes apparent.

Goko et al. [15] introduced a scheduling model with a metric state space that involves setup time and unknown execution time settings together. Their model includes a scheduling problem faced by repair companies where each worker needs to visit customers' houses, do the repair jobs, and then return to the office. The setup time corresponds to the shortest tour length for the customers' houses, and the execution time corresponds to the duration of the repair jobs. Their model also includes the online dial-a-ride problem [2, 4–7, 11, 22, 23], in which taxis are offered to pick up and drop passengers for transportation jobs. Our preemptive mUETS problem can be seen as an abstraction of their model, and hence, a similar approach can be used to solve it. However, the other problems (i.e., preemptive/non-preemptive sUETS and non-preemptive mUETS) are different and require other approaches.

The setting in which all jobs have an execution time of 0 and are released at time 0, with only the setup time considered as processing time, can be viewed as an offline load balancing problem. This problem is NP-hard even if the setup time is additive and the number of machines is two, as this special case corresponds to the PARTITION problem [14]. For the additive setup time, the list scheduling algorithm works as a $(2 - 1/m)$ -approximation algorithm. Moreover, this problem admits a polynomial time approximation scheme (PTAS) [19]. Svitkina and Fleischer [33] presented an $O(\sqrt{n/\log n})$ -approximation algorithm for the submodular setup times, and demonstrated that this is best possible. Furthermore, Nagano and Kishimoto [27] provided a $2 \cdot (\max_{j \in [m]} \frac{|S_j^*|}{1 + (|S_j^*| - 1)(1 - \kappa_c(S_j^*))})$ -approximation algorithm for the subadditive setup times, where (S_1^*, \dots, S_m^*) is an optimal partition and $\kappa_c(S)$ is the curvature of c at $S \subseteq J$.

2 Preliminaries

For a positive integer k , we write $[k]$ to denote the set $\{1, 2, \dots, k\}$. We are given a set J of n jobs and m identical machines. We assume that all jobs are given and released at time 0. As we will show in Section 2.2, this assumption only increases a constant factor in the competitive ratios. We denote the set of machines by $[m]$. We assume that $n \geq m \geq 2$, as the optimal scheduling is clear otherwise. The jobs are executed in batches, and each batch's processing time consists of two components: the *setup time* and the *execution time*.

The setup time is the total time one machine takes to set up jobs in the batch. For a batch $X \subseteq J$, the setup time for X is represented as $c(X) \in \mathbb{R}_+$. The execution time for each job $j \in J$, denoted by p_j , is unknown until its process is completed. We write $p(X)$ to denote the total execution time of a batch $X \subseteq J$, i.e., $p(X) = \sum_{j \in X} p_j$. Consequently, the overall processing time of a batch X on a single machine is given by $p(X) + c(X)$. We assume that the setup time $c(X)$ for every $X \subseteq J$ is available in advance, while the execution time of each job is unknown until its completion.

We also deal with a situation where one batch can be assigned to multiple machines. We assume that when a batch X is assigned to k machines, each of the machines incurs a setup time $c(X)$, and the jobs are executed on a first-come, first-served basis. For example, when a batch X is processed in a production system, we assume that each machine forms a queue of the attachments needed for jobs in X , and proceeds as follows: (i) dequeues and installs the first attachment in the queue, (ii) repeatedly executes an unprocessed job that is compatible with the current attachment as long as such a job exists, and (iii) removes the current attachment and returns to step (i) if the attachment queue is not empty. The minimum processing time for batch X across k machines falls within the range of $c(X) + p(X)/k$ and $c(X) + p(X)/k + \max_{j \in X} p_j$.

At each time, we can assign a batch of jobs that have not yet been completed and are not assigned anywhere else. In the sUETS and mUETS problems, a batch can be allocated to a single machine and multiple machines, respectively. Our objective is to minimize the makespan, which is the time at which all the jobs are completed.

We assume that the setup time function $c: 2^J \rightarrow \mathbb{R}_+$ is *monotone subadditive*. Monotonicity means that $c(X) \leq c(Y)$ for any $X \subseteq Y \subseteq J$. Subadditivity means that for any disjoint subsets X and Y from J , the sum of their setup times is greater than or equal to the setup time of their union, i.e., $c(X) + c(Y) \geq c(X \cup Y)$. Note that this assumption of subadditivity does not lose generality. To observe this, for any (not necessarily subadditive) function c , define the function $\bar{c}(X) = \min_{(X_1, \dots, X_\ell): \text{partition of } X} \sum_{i=1}^{\ell} c(X_i)$. This new function \bar{c} is monotone and subadditive by construction – it represents the minimum total setup time

achievable if one were allowed to split the batch X into sub-batches and incur the setup cost separately for each. Importantly, replacing c with \bar{c} in our analysis does not change the fundamental nature of the scheduling problem, and thus we assume without loss of generality that the setup time function is monotone and subadditive.

We explore two settings based on whether an allocated batch can be preempted. In the non-preemptive setting, once a batch is assigned, it must be processed to completion without interruption. Conversely, in the preemptive setting, the processing of a batch can be halted and canceled. Suppose that a batch X is assigned to k machines and is preempted after t units of time from the start of its processing. In this case, we assume that the machines will become available to process another batch after at most $c(X) + \max_{j \in X} p_j$ units of time. This assumption ensures that the processing time does not increase even if preemption occurs during the setup phase. Furthermore, at the moment of preemption, let $X' \subseteq X$ denote the set of jobs that have been completed. Then, the following inequality holds: $c(X) + p(X')/k + \max_{j \in X} p_j \geq t$. This condition ensures that the total time spent up to the point of preemption is consistent with the setup and execution times for the completed jobs.

Note that, in any setting, we can assume that an optimal schedule does not perform preemptions and assigns only one batch to each machine at the beginning because the setup time c is subadditive. Thus, an optimal schedule can be represented as a partition of jobs (X_1^*, \dots, X_m^*) and the optimal makespan is $\max_{i \in [m]} (c(X_i^*) + p(X_i^*))$ independently of the settings. Based on this observation, we derive a lower bound for the optimal makespan, as formalized in the following lemma.

► **Lemma 1.** *The optimal makespan is at least*

$$\max \left\{ \min_{(X_1, \dots, X_m): \text{partition of } J} \max_{i \in [m]} c(X_i), (c(J) + p(J))/m, \max_{j \in J} (c(\{j\}) + p_j) \right\}.$$

Proof. Let (X_1^*, \dots, X_m^*) be an optimal schedule. Then, the optimal makespan is at least

$$\max_{i \in [m]} (c(X_i^*) + p(X_i^*)) \geq \max_{i \in [m]} c(X_i^*) \geq \min_{(X_1, \dots, X_m): \text{partition of } J} \max_{i \in [m]} c(X_i).$$

Additionally, we have

$$\max_{i \in [m]} (c(X_i^*) + p(X_i^*)) \geq \max_{j \in J} (c(\{j\}) + p_j) \geq (c(J) + p(J))/m$$

by monotone subadditivity of the setup time. Therefore, we have the lower bound. ◀

2.1 Examples of setup times

This subsection illustrates several examples of setup times raised in practical applications. In addition, we discuss the approximability of the partition (load balancing) problem

$$\min_{(X_1, \dots, X_m): \text{partition of } J} \max_{i \in [m]} c(X_i) \tag{1}$$

for each class of setup times.

Constant setup times. One of the simplest examples of the setup time family is the constant setup time, where the setup time is a certain constant independent of the batch. Formally, the setup time is represented as $c(X) = 1$ for $X \in 2^J \setminus \{\emptyset\}$ and $c(\emptyset) = 0$. It is not difficult to see that any partition of the jobs (X_1, \dots, X_m) is optimal for the problem (1).

Type-specific setup times. In production systems with attachments, jobs are partitioned based on their types. Let T be the set of types and $(J_t)_{t \in T}$ is the partition of jobs with types T , i.e., $\bigcup_{t \in T} J_t = J$ and $J_t \cap J_{t'} = \emptyset$ for all distinct $t, t' \in T$. Let w_t be the setup time for type t . Then, the setup time for $X \subseteq J$ is defined as $c(X) = \sum_{t \in T: X \cap J_t \neq \emptyset} w_t$. Additionally, we say that a type-specific setup time c is unweighted if $w_t = 1$ for all $t \in T$, i.e., $c(X) = |\{t \in T : X \cap J_t \neq \emptyset\}|$ for $X \subseteq J$. We can obtain a PTAS for the problem (1) with type-specific setup times by treating each J_t as a single job and applying a PTAS for the load balancing problem [19]. Moreover, for the unweighted case, the problem (1) is solvable in polynomial time because an optimal solution can be obtained by simply balancing the number of types assigned to each machine.

Library-based setup times. In cloud computing environments, where each job requires the installation of multiple libraries, and each library takes a fixed amount of time to install. The setup time needed to process a batch of jobs is the total installation time of the required libraries. Let L be the set of libraries, and let $L_j \subseteq L$ be the set of required libraries for job $j \in J$. Let w_ℓ be the installation time for library $\ell \in L$. Then, the setup time for $X \subseteq J$ is defined as $c(X) = \sum_{\ell \in \bigcup_{j \in X} L_j} w_\ell$. Note that if every job $j \in J$ requires a unique library (i.e., $|L_j| = 1$), this is a type-specific setup time. This setup time is a monotone submodular function; more specifically, it is a weighted coverage function. Hence, we can apply the $O(\sqrt{n/\log n})$ -approximation algorithm for the uniform submodular load balancing problem [33] to solve the problem (1).

TSP-based setup times. For applications like repair companies, the setup time is defined by the optimal value of a traveling salesman problem (TSP) instance. Let (V, d) be a metric space with an origin $o \in V$. The origin corresponds to the location of the company. Each job $j \in J$ is associated with a point $v_j \in V$. Then, the setup time for $X \subseteq J$ is defined as the optimal value of the TSP on $V' = \{v_j : j \in X\} \cup \{o\}$. Note that this class of setup times is a generalization of type-specific setup times. Indeed, for a star metric $(T \cup \{o\}, d)$ where $d(t, o) = w_t/2$ for all $t \in T$ and $d(t, t') = (w_t + w_{t'})/2$ for all distinct $t, t' \in T$, the setup time is $c(X) = \sum_{t \in T: X \cap J_t \neq \emptyset} w_t$ for $X \subseteq J$ when $v_j = t$ for each job $j \in J_t$. For TSP-based setup times, the problem (1) can be viewed as the m -TSP. It is known that the m -TSP admits a 2.5-approximation algorithm [8, 12].

2.2 Reduction of the Problem with Release Time

In this subsection, we show that our assumption that all jobs have a release time of 0 (i.e., each job is available for processing from time 0) does not lose generality. Suppose instead that each job has a release time, and its existence is hidden until its release time. Let ALG be a ρ -competitive algorithm for a UETS problem in a certain setting without release times. Then, we demonstrate that ALG can be transformed into a $(2\rho + 1)$ -competitive algorithm for the UETS problem with release times of the corresponding setting. For this end, we use the *IGNORE* strategy, which appeared in the paper of Shmoys et al. [32] and is named by Ascheuer et al. [2].

The *IGNORE* strategy keeps the machines remain idle until a set S of jobs appears. Then, *IGNORE* immediately decides a schedule for the jobs in S following the algorithm ALG and assigns job batches to machines. We refer to this schedule as a subschedule for S . All jobs that arrive during the process of the subschedule are temporarily ignored until the subschedule for S is completed. After all the machines complete the jobs in S , *IGNORE* decides a subschedule for all new jobs that arrived during the previous process. If there are no such jobs, all the machines become idle. The *IGNORE* strategy repeats this procedure.

► **Theorem 2.** *The IGNORE strategy with a ρ -competitive algorithm ALG is $(2\rho + 1)$ -competitive for the UETS problem with release time.*

Proof. Fix an instance, and let OPT be the optimal makespan. For a subset of jobs J' , we denote $\text{ALG}(J')$ as the makespan of the algorithm, assuming that jobs in J' are given at time 0. Note that $\text{ALG}(J') \leq \rho \cdot \text{OPT}$ by the monotonicity of the optimal makespan with respect to the jobs.

If the instance contains no job, then IGNORE is clearly optimal. Thus, we assume that the instance contains at least one job. Let t^* be the last release time of the jobs. Then, $\text{OPT} \geq t^*$ as the last released job can only be processed from t^* .

Suppose that the machines are idle at time t^* . Let R be the set of jobs processed in the last subschedule by the IGNORE strategy. Then, the makespan of IGNORE is

$$t^* + \text{ALG}(R) \leq \text{OPT} + \rho \cdot \text{OPT} \leq (2\rho + 1) \cdot \text{OPT}.$$

On the other hand, suppose that some machines are not idle at time t^* . Then t^* is in the second last subschedule of the algorithm, and the last subschedule starts right after the second-to-last subschedule ends. Let R and S be the sets of jobs processed in the last and the second-to-last subschedules, respectively. Then, the makespan of IGNORE is

$$t^* + \text{ALG}(S) + \text{ALG}(R) \leq \text{OPT} + \rho \cdot \text{OPT} + \rho \cdot \text{OPT} = (2\rho + 1) \cdot \text{OPT}.$$

Therefore, the competitive ratio of IGNORE is at most $2\rho + 1$. ◀

It should be noted that this reduction remains valid even when the competitive ratio ρ of ALG depends on the number of machines m or the number of jobs n , provided it is monotonically nondecreasing with respect to n . From this theorem, the optimum competitive ratio is the same up to a constant factor regardless of the release time.

3 Single Machine Batch Allocation

In this section, we analyze the sUETS problem where each batch can be allocated only on a single machine. Recall that n is the number of jobs and m is the number of machines.

3.1 Algorithms

We first observe that the algorithm of processing the batch consisting of all jobs by a single machine without preemption is m -competitive, where m is the number of machines. Note that this algorithm is feasible for any settings of sUETS and mUETS since no cancellation is performed.

► **Theorem 3.** *The algorithm of processing the batch consisting of all jobs by a single machine is m -competitive for the non-preemptive sUETS problem.*

Proof. As the makespan of the algorithm is $c(J) + p(J)$ and the optimal makespan is at least $(C(J) + p(J))/m$ by Lemma 1, the competitive ratio is at most

$$\frac{c(J) + p(J)}{\max_{i \in [m]} (c(X_i^*) + p(X_i^*))} \leq \frac{c(J) + p(J)}{\sum_{i \in [m]} (c(X_i^*) + p(X_i^*)) / m} \leq \frac{c(J) + p(J)}{(c(J) + p(J)) / m} = m,$$

where the second inequality is implied by the subadditivity of the setup time c . ◀

Second, we analyze the competitive ratio of the list scheduling algorithm.

► **Theorem 4.** *The algorithm that greedily assigns a batch consisting of a single unprocessed job to any available single machine is $O(n/m)$ -competitive for the non-preemptive sUETS problem.*

Proof. The makespan of the algorithm is at most

$$\frac{1}{m} \sum_{j \in J} (c(\{j\}) + p_j) + \max_{j \in J} (c(\{j\}) + p_j) \leq \frac{n \cdot \text{OPT}}{m} + \text{OPT} = O(n/m) \cdot \text{OPT},$$

where the inequality holds by Lemma 1. Thus, the competitive ratio is at most $O(n/m)$. ◀

Next, we provide an $O(\sqrt{n/m})$ -competitive algorithm, which is an improved version of the list scheduling algorithm. The idea is to reduce the total setup time by grouping jobs. The algorithm first divides the jobs into $m + \lceil \sqrt{mn} \rceil$ batches in such a way that this partition minimizes the maximum setup time, under the condition that each batch contains at most $\lceil \sqrt{n/m} \rceil$ jobs. Then, it processes the batches according to the list scheduling algorithm, which greedily allocates an unprocessed batch to any available machine. The formal description of this algorithm is summarized in Algorithm 1.

■ **Algorithm 1** $O(\sqrt{n/m})$ -competitive algorithm for the non-preemptive sUETS problem.

-
- 1 Let $k = m + \lceil \sqrt{mn} \rceil$;
 - 2 Compute a k -partition (X_1, \dots, X_k) of the jobs J such that $|X_i| \leq \lceil \sqrt{n/m} \rceil$ for all $i \in [k]$, and among such partitions, minimize $\max_{i \in [k]} c(X_i)$;
 - 3 **for** $i \leftarrow 1, 2, \dots, k$ **do**
 - 4 Wait until at least one machine is available;
 - 5 Assign batch X_i to an available machine;
-

► **Theorem 5.** *Algorithm 1 is $O(\sqrt{n/m})$ -competitive for the non-preemptive sUETS problem.*

Proof. Let (X_1^*, \dots, X_m^*) be the optimal schedule and let OPT denote the optimal makespan $\max_{i \in [m]} (c(X_i^*) + p(X_i^*))$. Define $k = m + \lceil \sqrt{mn} \rceil$, and let (X_1, \dots, X_k) be a k -partition of the jobs J that minimizes $\max_{i \in [k]} c(X_i)$ subject to $|X_i| \leq \lceil \sqrt{n/m} \rceil$ for all $i \in [k]$.

We first observe that $\max_{i \in [k]} c(X_i) \leq \text{OPT}$. This is because by dividing each X_i^* with $i \in [m]$ into $\lceil |X_i^*| / \sqrt{n/m} \rceil$ sub-batches of almost equal size, we can obtain a partition (Y_1, \dots, Y_k) that refines (X_1^*, \dots, X_m^*) . Here, we have $|Y_{i'}| \leq \max_{i \in [m]} \lceil |X_i^*| / \lceil |X_i^*| / \sqrt{n/m} \rceil \rceil \leq \lceil \sqrt{n/m} \rceil$ for all $i' \in [k]$. Thus, it holds that $\max_{i \in [k]} c(X_i) \leq \max_{i \in [k]} c(Y_i) \leq \max_{i \in [m]} c(X_i^*) \leq \text{OPT}$.

Then, the makespan of the algorithm is at most

$$\begin{aligned} & \frac{1}{m} \sum_{i \in [k]} (c(X_i) + p(X_i)) + \max_{i \in [k]} (c(X_i) + p(X_i)) \\ & \leq \frac{k \cdot \text{OPT} + p(J)}{m} + \max_{i \in [k]} |X_i| \cdot \max_{j \in J} (c(\{j\}) + p_j) \\ & \leq \left(k/m + 1 + \lceil \sqrt{n/m} \rceil \right) \cdot \text{OPT} = O(\sqrt{n/m}) \cdot \text{OPT}, \end{aligned}$$

where the first inequality follows from the subadditivity of the setup time and the second inequality from Lemma 1. Hence, Algorithm 1 is $O(\sqrt{n/m})$ -competitive. ◀

By combining Theorems 3 and 5, we obtain the following corollary.

► **Corollary 6.** *There exists an $O(n^{1/3})$ -competitive algorithm for the non-preemptive sUETS problem.*

Proof. If $m \leq n^{1/3}$, then the algorithm that processes the batch J by a single machine is $m = O(n^{1/3})$ -competitive by Theorem 3. On the other hand, if $m > n^{1/3}$, the competitive ratio of Algorithm 1 is $O(\sqrt{n/m}) = O(n^{1/3})$ by Theorem 5. Thus, an $O(n^{1/3})$ -competitive algorithm exists for either case. ◀

► **Remark 7.** Obtaining the partition at line 2 in Algorithm 1 is computationally hard. Thus, we mention the competitive ratio when we can utilize an α -approximation algorithm for the problem (1). Let (X'_1, \dots, X'_m) be an m -partition of the jobs J that is obtained by the approximation algorithm. Let $k' = m + \lceil \sqrt{mn/\alpha} \rceil$. Then, we can easily construct a k' -partition $(Y'_1, \dots, Y'_{k'})$ that is a refinement of (X'_1, \dots, X'_m) and satisfies $|Y'_i| \leq \lceil \sqrt{\alpha \cdot n/m} \rceil$ for all $i \in [k']$. By using k' instead of k and employing this partition at line 2, Algorithm 1 is $O(\alpha + \sqrt{\alpha \cdot n/m})$ -competitive for the non-preemptive sUETS problem. By combining this with Theorem 4, we can obtain an $O(\min\{\sqrt{\alpha \cdot n/m}, n/m\})$ -competitive algorithm. Moreover, by combining this with Theorem 3, we can also obtain an $O(\min\{(\alpha n)^{1/3}, \sqrt{n}\})$ -competitive algorithm.

In the rest of this subsection, we provide an $O(\log n / \log \log n)$ -competitive algorithm for the preemptive sUETS problem. For a given instance of the sUETS problem, let q be an integer such that $q^q \geq n > (q-1)^{q-1}$. Note that $q = \Theta(\log n / \log \log n)$. The execution of our algorithm consists of $q+1$ phases. Without loss of generality, we may assume that $m \geq 2q$, since otherwise the competitive ratio of the algorithm in Theorem 3 is $m = O(q) = O(\log n / \log \log n)$.

The intuition of our algorithm is as follows. Our algorithm repeatedly completes a $(1 - 1/q)$ -fraction of the jobs in each phase. Consequently, all the jobs can be processed in $O(q)$ iterations. This increases the cost of the setup time by a factor of $O(q)$. Additionally, we ensure that at least m/q machines work at any point of the algorithm. This guarantees that the cost in terms of execution time will only increase by a factor of at most q . By these properties, the competitive ratio of the algorithm is $O(q) = O(\log n / \log \log n)$.

Let us explain our algorithm more precisely. Initially, it computes an m -partition (X_1, \dots, X_m) of the jobs J that minimizes the maximum setup time $\max_{i \in [m]} c(X_i)$. In the first phase, X_i is allocated to machine i for each $i \in [m]$. Then, each machine processes the assigned batch until either the machine completes the batch or the number of uncompleted machines becomes less than or equal to $\lfloor m/q \rfloor$. At that time, the machines still processing will preempt their batches. In the k th phase ($k = 2, 3, \dots, q$), the algorithm computes an m -partition $(Y_1^{(k)}, \dots, Y_m^{(k)})$ of the remaining jobs such that, for any $i \in [m]$, $|Y_i^{(k)}| \leq n/q^{k-1}$ and $Y_i^{(k)} \subseteq X_j$ for some $j \in [m]$. We can always obtain such a partition by dividing each batch left in the previous phase into q sub-batches. Thereafter, each machine processes the assigned batch until either the batch is completed or the number of uncompleted machines becomes less than or equal to $\lfloor m/q \rfloor$. At the end of q th phase, the number of remaining jobs is at most $(n/q^{q-1}) \cdot \lfloor m/q \rfloor \leq m \cdot (n/q^q) \leq m$. In the $(q+1)$ st phase, the algorithm assigns up to one remaining job to each machine. A formal description of the algorithm is provided in Algorithm 2.

► **Theorem 8.** *Algorithm 2 is $O(\frac{\log n}{\log \log n})$ -competitive for the preemptive sUETS problem.*

Proof. Let q be an integer such that $q^q \geq n > (q-1)^{q-1}$. If $m < 2q$, then the competitive ratio is $m = O(q) = O(\log n / \log \log n)$ by Theorem 3. Thus, we assume $m \geq 2q$.

■ **Algorithm 2** $O\left(\frac{\log n}{\log \log n}\right)$ -competitive algorithm for the preemptive sUETS problem.

```

1 Let  $q$  be an integer such that  $q^q \geq n > (q-1)^{q-1}$ ;
2 if  $m < 2q$  then Assign batch  $J$  to one machine and exit;
3 Compute an  $m$ -partition  $(X_1, \dots, X_m)$  of the jobs  $J$  that minimizes  $\max_{i \in [m]} c(X_i)$ ;
4 Let  $R^{(0)} \leftarrow J$  be the set of uncompleted jobs;
5 for  $k \leftarrow 1, 2, \dots, q$  do // phase  $k$ 
6   Compute an  $m$ -partition of  $(Y_1^{(k)}, \dots, Y_m^{(k)})$  of the remaining jobs  $R^{(k-1)}$  such
   that, for each  $i \in [m]$ ,  $|Y_i^{(k)}| \leq n/q^{k-1}$  and  $Y_i^{(k)} \subseteq X_j$  for some  $j \in [m]$ ;
7   Assign batch  $Y_i^{(k)}$  to machine  $i$  for each  $i \in [m]$ ;
8   Each machine processes the assigned batch until the batch is completed or the
   number of uncompleted machines becomes less than or equal to  $\lfloor m/q \rfloor$ ;
9   Let  $R^{(k)}$  be the set of uncompleted jobs;
10 Assign up to one job  $R^{(q)}$  per machine; // phase  $q+1$ 

```

Let OPT be the optimal makespan and let $p_{\max} = \max_{j \in J} p_j$. In addition, let τ_k be the time length of phase $k \in [q+1]$. By the definition of the algorithm, the jobs in $R^{(k-1)} \setminus R^{(k)}$ are completed in phase $k \in [q]$. Thus, the total execution time of jobs in phase $k \in [q]$ is at most $p(R^{(k-1)} \setminus R^{(k)}) + p_{\max} \cdot \lfloor m/q \rfloor$ because at most $\lfloor m/q \rfloor$ jobs are partially executed. Taking into account the setup time of at most $\max_{i \in [m]} c(X_i)$ and the time for preemption of at most $\max_{i \in [m]} c(X_i) + p_{\max}$, the length of phase $k \in [q]$ is

$$\begin{aligned} \tau_k &\leq \max_{i \in [m]} c(X_i) + \frac{p(R^{(k-1)} \setminus R^{(k)}) + p_{\max} \cdot \lfloor m/q \rfloor}{\lfloor m/q \rfloor} + \left(\max_{i \in [m]} c(X_i) + p_{\max} \right) \\ &\leq \frac{q}{m-q} \cdot p(R^{(k-1)} \setminus R^{(k)}) + 4\text{OPT} \leq \frac{2q}{m} \cdot p(R^{(k-1)} \setminus R^{(k)}) + 4\text{OPT}, \end{aligned}$$

where the second inequality is by Lemma 1 and the third is by $m \geq 2q$. In addition, by Lemma 1, the time length τ_{q+1} of phase $q+1$ is

$$\tau_{q+1} \leq \max_{j \in J} (c(\{j\}) + p_j) \leq \text{OPT}.$$

Thus, the makespan of Algorithm 2 is

$$\begin{aligned} \sum_{k=1}^{q+1} \tau_k &\leq (4q+1)\text{OPT} + \frac{2q}{m} \cdot \sum_{k=1}^q p(R^{(k-1)} \setminus R^{(k)}) \\ &\leq (4q+1)\text{OPT} + \frac{2q}{m} \cdot p(J) \leq (4q+1)\text{OPT} + 2q \cdot \text{OPT} = O(q) \cdot \text{OPT}. \end{aligned}$$

Hence, the competitive ratio of Algorithm 2 is $O(q) = O(\log n / \log \log n)$. ◀

► **Remark 9.** Similar to Remark 7, suppose that we take an m -partition (X_1, \dots, X_m) of the jobs J that is an α -approximation of the problem (1) at line 6 in Algorithm 2. Then, the competitive ratio of Algorithm 2 becomes $O(\alpha \cdot \frac{\log n}{\log \log n})$ -competitive for the preemptive sUETS problem. Moreover, by setting q to be the integer that satisfies $q^q \geq n^\alpha > (q-1)^{q-1}$, we can slightly improve the competitive ratio to $O(\frac{\log n^\alpha}{\log \log n^\alpha})$.

■ **Algorithm 3** $O(\sqrt{m})$ -competitive algorithm for the non-preemptive mUETS problem.

-
- 1 Let $k = \lfloor \sqrt{m} \rfloor$;
 - 2 Compute a k -partition (X_1, \dots, X_k) of the jobs J that minimize $\max_{i \in [k]} c(X_i)$;
 - 3 Assign batch X_i to $\lfloor m/k \rfloor$ machines for each $i \in [k]$;
-

3.2 Lower bounds

In this subsection, we show that the algorithms provided in Section 3.1 are asymptotically optimal with respect to both the number of machines and the number of jobs.

We first provide a lower bound for the non-preemptive setting.

► **Theorem 10.** *The competitive ratio of the non-preemptive sUETS problem is at least $\Omega(m)$ and $\Omega(n^{1/3})$ even for constant setup times.*

Proof. Let $n = m^3$ and $c(X) = 1$ for any non-empty batch $X \subseteq J$ (i.e., constant setup times). We will set the execution time 1 for at most m jobs (referred to as *heavy jobs*) and 0 for the other jobs (referred to as *light jobs*) depending on the behavior of a given online algorithm. It is not difficult to see that the optimal makespan is at most 2.

We fix an online algorithm. Let us consider its behavior when every job is light. Suppose that every batch assigned to machines consists of at most m jobs. Then, the algorithm assigns at least $n/m = m^2$ batches to the machines in total. Consequently, some machines must process at least m batches, which requires m setups. Thus, the makespan is at least m in this case, and we have done.

Conversely, suppose that the algorithm allocates a batch consisting of more than m jobs in the instance where every job is light. In another instance where m jobs in the first such batch are set to be heavy, the algorithm's behavior is the same until the batch is assigned. Thus, the algorithm assigns all the heavy jobs to one machine, and the makespan is at least m . Therefore, in both cases, the competitive ratio is at least $\Omega(m) = \Omega(n^{1/3})$. ◀

Next, we provide a lower bound for the preemptive setting.

► **Theorem 11.** *The competitive ratio of the preemptive sUETS problem is at least $\Omega(m)$ and $\Omega(\log n / \log \log n)$ even for constant setup times.*

4 Multiple Machines Batch Allocation

In this section, we explore the mUETS problem, where a batch can be allocated to multiple machines.

4.1 Algorithms

We first provide an $O(\sqrt{m})$ -competitive algorithm for the non-preemptive mUETS problem. The algorithm computes a $\lfloor \sqrt{m} \rfloor$ -partition of J that minimizes maximum setup time. Then, it allocates each batch to $\lfloor \sqrt{m} \rfloor$ machines. The algorithm is formally described in Algorithm 3.

► **Theorem 12.** *Algorithm 3 is $O(\sqrt{m})$ -competitive for the non-preemptive mUETS problem.*

■ **Algorithm 4** $O\left(\frac{\log m}{\log \log m}\right)$ -competitive algorithm for the preemptive mUETS problem.

-
- 1 Let q be an integer such that $q^q \geq m > (q-1)^{q-1}$;
 - 2 Compute an m -partition X_1, \dots, X_m of the jobs J that minimizes $\max_{i \in [m]} c(X_i)$;
 - 3 Set $k^* \leftarrow \lfloor \log_q m \rfloor + 1$ and let $I^{(0)} \leftarrow [m]$ be the set of unprocessed batch indices;
 - 4 **for** $k \leftarrow 1, 2, \dots, k^*$ **do** // **phase** k
 - 5 Assign batch X_i to q^{k-1} machines for each $i \in I^{(k-1)}$;
 - 6 Continue processing the assignment until the number of uncompleted batches becomes $\lfloor m/q^k \rfloor$. Once this condition is met, preempt all remaining uncompleted batches;
 - 7 Define $I^{(k)}$ to be the set of uncompleted batch indices;
-

Proof. Let (X_1^*, \dots, X_m^*) be an optimal schedule and let OPT be the optimal makespan. In addition, let $k = \lfloor \sqrt{m} \rfloor$ and (X_1, \dots, X_k) be a k -partition of J that minimizes $\max_{i \in [k]} c(X_i)$. We analyze the algorithm that allocates each batch X_i to $\lfloor m/k \rfloor$ machines. The makespan of the algorithm is at most $\max_{i \in [k]} \left(c(X_i) + \frac{p(X_i)}{\lfloor m/k \rfloor} + \max_{j \in X_i} p_j \right)$. By the choice of (X_1, \dots, X_k) and the subadditivity of the setup time c , we have

$$\begin{aligned} \max_{i \in [k]} c(X_i) &\leq \max_{i \in [k]} c \left(\bigcup_{j=1}^{\lfloor m/k \rfloor} X_{\lfloor m/k \rfloor \cdot (i-1) + j}^* \right) \\ &\leq \max_{i \in [k]} \sum_{j=1}^{\lfloor m/k \rfloor} c(X_{\lfloor m/k \rfloor \cdot (i-1) + j}^*) \leq \left\lceil \frac{m}{k} \right\rceil \cdot \text{OPT}, \end{aligned} \quad (2)$$

where we denote $X_i^* = \emptyset$ for $i > k$. Therefore, by Lemma 1 and (2), the makespan is at most

$$\max_{i \in [k]} \left(c(X_i) + \frac{p(X_i)}{\lfloor m/k \rfloor} + \max_{j \in X_i} p_j \right) \leq \left(\left\lceil \frac{m}{k} \right\rceil + \frac{m}{\lfloor m/k \rfloor} + 1 \right) \cdot \text{OPT} = O(\sqrt{m}) \cdot \text{OPT},$$

which means that the competitive ratio of the algorithm is $O(\sqrt{m})$. ◀

By combining Theorems 5 and 12, we can obtain the following corollary.

► **Corollary 13.** *There is an $O(n^{1/4})$ -competitive algorithm for the non-preemptive mUETS problem.*

Proof. If $m \leq \sqrt{n}$, then the algorithm in Theorem 12 is $O(\sqrt{m}) = O(n^{1/4})$ -competitive. On the other hand, if $m > \sqrt{n}$, the competitive ratio of Theorem 5 is $O(\sqrt{n/m}) = O(n^{1/4})$. Thus, in either case, there is an $O(n^{1/4})$ -competitive algorithm. ◀

► **Remark 14.** To account for the computational issue, suppose that we only have an m -partition (X'_1, \dots, X'_m) of the jobs J that is an α -approximation for the problem (1). Let $k' = \lfloor \sqrt{\alpha m} \rfloor$. Then, the schedule that assigns batch $\bigcup_{j=1}^{\lfloor m/k' \rfloor} X'_{\lfloor m/k' \rfloor \cdot (i-1) + j}$ to $\lfloor m/k' \rfloor$ machines for each $i \in [k']$ is $O(\sqrt{\alpha m})$ -competitive for the non-preemptive mUETS problem. By combining this with Remark 7, we can obtain an $O(\min\{\sqrt{\alpha} \cdot n^{1/4}, \sqrt{n}\})$ -competitive algorithm for the non-preemptive mUETS problem.

For the preemptive mUETS problem, Algorithm 2 is also $O(\log n / \log \log n)$ -competitive, and this is asymptotically best possible as we will see in Theorem 18. Thus, even when we are allowed to assign a job batch to more than one machine, we cannot improve the competitive

ratio with respect to the number n of jobs. In contrast, we show that the competitive ratio with respect to the number m of machines can be exponentially improved by allowing batches to be assigned to multiple machines.

Let q be an integer such that $q^q > m \geq (q-1)^{q-1}$ and let k^* be $\lfloor \log_q m \rfloor + 1$. Here, $k^* \leq q$ and $q^{k^*-1} \leq m < q^{k^*}$. We have $q \geq 2$ from the assumption that $m \geq 2$.

Our algorithm for the preemptive mUETS is similar to Algorithm 2 for the preemptive sUETS. However, instead of splitting the uncompleted batch into several smaller batches of similar size, the algorithm increases the number of machines assigned to that batch. This ensures that the competitive ratio of the algorithm does not depend on the number of jobs n .

Our algorithm computes an m -partition (X_1, \dots, X_m) of the jobs J that minimizes the maximum setup time $\max_{i \in [m]} c(X_i)$. In the first phase, each batch X_i is allocated to one machine. Then, each batch is processed until it is completed, or the number of uncompleted batches becomes less than or equal to $\lfloor m/q \rfloor$. In the k th phase ($k = 2, 3, \dots, k^*$), each uncompleted batch X_i is assigned to q^{k-1} machines, and it is processed until it is completed, or the number of uncompleted batches becomes $\lfloor m/q^k \rfloor$. If the number of uncompleted batches becomes smaller than $\lfloor m/q^k \rfloor$ because multiple batches are completed simultaneously, then break the tie arbitrarily and defer the rest batches to the next phase. Note that this allocation is feasible because $\lfloor m/q^{k-1} \rfloor \cdot q^{k-1} \leq m$. At the end of the k^* th phase, all the batches are completed since $\lfloor m/q^{k^*} \rfloor = 0$ by $k^* = \lfloor \log_q m \rfloor + 1 > \log_q m$. Our algorithm is formally described in Algorithm 4. It should be noted that this algorithm is based on a similar idea to that of [15, Algorithm 1].

► **Theorem 15.** *Algorithm 4 is $O(\frac{\log m}{\log \log m})$ -competitive for the preemptive mUETS problem.*

Proof. Let OPT be the optimal makespan and let $p_{\max} = \max_{j \in J} p_j$. For each $k \in [k^*]$, let $S^{(k)} = I^{(k-1)} \setminus I^{(k)}$ be the set of indices of batches that have been completed in the k th phase. By the definition of the algorithm, we have

$$|S^{(k)}| = \left\lfloor \frac{m}{q^{k-1}} \right\rfloor - \left\lfloor \frac{m}{q^k} \right\rfloor \geq \left\lfloor \frac{m}{q^{k-1}} \right\rfloor - \frac{1}{q} \cdot \left\lfloor \frac{m}{q^{k-1}} \right\rfloor = \left(1 - \frac{1}{q}\right) \left\lfloor \frac{m}{q^{k-1}} \right\rfloor \geq \frac{1}{2} \cdot \left\lfloor \frac{m}{q^{k-1}} \right\rfloor.$$

Let τ_k be the time length of phase $k \in [k^*]$. We first bound it for $k \in [k^* - 1]$. As every batch in $S^{(k+1)}$ is not finished in phase k and the preemption takes time at most $\max_{i \in S^{(k)}} (c(X_i) + \max_{j \in X_i} p_j) \leq 2\text{OPT}$, we have

$$\begin{aligned} \tau_k &\leq \min_{i \in S^{(k+1)}} \left(c(X_i) + \frac{p(X_i)}{q^{k-1}} + p_{\max} \right) + 2\text{OPT} \\ &\leq \frac{1}{|S^{(k+1)}|} \cdot \sum_{i \in S^{(k+1)}} \left(c(X_i) + \frac{p(X_i)}{q^{k-1}} + p_{\max} \right) + 2\text{OPT} \\ &\leq \sum_{i \in S^{(k+1)}} \frac{1}{|S^{(k+1)}|} \cdot \frac{p(X_i)}{q^{k-1}} + 4\text{OPT} \leq \frac{m/q^k}{\frac{1}{2} \cdot \lfloor m/q^k \rfloor} \cdot \sum_{i \in S^{(k+1)}} q \cdot \frac{p(X_i)}{m} + 4\text{OPT}. \end{aligned}$$

Note that $x/\lfloor x \rfloor \leq 2$ if $x \geq 1$ and $m/q^k \geq 1$ for $k \in [k^* - 1]$. Thus, we have

$$\tau_k \leq \sum_{i \in S^{(k+1)}} 4q \cdot \frac{p(X_i)}{m} + 4\text{OPT} \tag{3}$$

for every $k \in [k^* - 1]$.

Next, we bound the time length τ_{k^*} of phase k^* . As the batch X_i for each $i \in S^{(k^*)}$ is processed by q^{k^*-1} machines, we have

$$\begin{aligned}\tau_{k^*} &\leq \max_{i \in S^{(k^*)}} \left(c(X_i) + \frac{p(X_i)}{q^{k^*-1}} + p_{\max} \right) \\ &\leq \frac{p(J)}{q^{k^*-1}} + 2\text{OPT} < q \cdot \frac{p(J)}{m} + 2\text{OPT} \leq (q+2)\text{OPT}.\end{aligned}\tag{4}$$

Hence, by (3) and (4), the makespan of the algorithm is

$$\begin{aligned}\sum_{k=1}^{k^*} \tau_k &\leq \sum_{k=1}^{k^*-1} \left(\sum_{i \in S^{(k+1)}} 4q \cdot \frac{p(X_i)}{m} + 4\text{OPT} \right) + (q+2)\text{OPT} \\ &\leq 4q \cdot \frac{p(J)}{m} + 4(q-1)\text{OPT} + (q+2)\text{OPT} \leq (9q-2)\text{OPT} = O(q) \cdot \text{OPT}.\end{aligned}$$

Therefore, the competitive ratio of Algorithm 4 is $O(q) = O(\log m / \log \log m)$. \blacktriangleleft

► **Remark 16.** Suppose that we take an m -partition (X_1, \dots, X_m) of the jobs J that is an α -approximation for the problem (1) at line 2 in Algorithm 2. Then, the competitive ratio of Algorithm 4 becomes $O(\alpha \cdot \frac{\log m}{\log \log m})$ -competitive for the preemptive mUETS problem. Moreover, by setting q to be the integer that satisfies $q^q \geq m^\alpha > (q-1)^{q-1}$, we can slightly improve the competitive ratio to $O(\frac{\log m^\alpha}{\log \log m^\alpha})$.

4.2 Lower Bounds

In this subsection, we show the asymptotic optimality of our algorithms. We first provide lower bounds for the non-preemptive case.

► **Theorem 17.** *The competitive ratio of the non-preemptive mUETS is at least $\Omega(\sqrt{m})$ and $\Omega(n^{1/4})$ even for unweighted type-specific setup times.*

Next, we provide lower bounds for the preemptive case.

► **Theorem 18.** *The competitive ratio of the preemptive mUETS problem is at least $\Omega(\log m / \log \log m)$ and $\Omega(\log n / \log \log n)$ even for unweighted type-specific setup times.*

5 Conclusion and Discussion

In this paper, we introduced the UETS problem and studied it in terms of competitive analysis. We obtained tight bounds of the competitive ratio in each setting. In the following, we discuss the application of our results to variant settings.

When a batch $X \subseteq J$ is assigned to k machines, we have assumed that every machine incurs a setup time $c(X)$. However, for instance, in the production system example, unnecessary reassignment of attachments can be skipped. Additionally, if a machine processes multiple batches, the setup time incurred from the second or subsequent batch may be reduced. Our algorithmic results remain valid even for these cases as long as an optimal schedule assigns only one batch to each machine separately. This is because, for algorithms, a decrease in setup time only contributes to decreasing the makespan.

In the preemptive setting, we assumed that a job completed at the time of preemption does not need to be executed again. We can also consider a setting in which, if a batch process is preempted, the entire batch must be restarted from scratch. This means that even

completed jobs in the batch must be executed again. Algorithm 4 also works in this setting, and it is $O(\log m / \log \log m)$ -competitive because its analysis does not use the information of which jobs are completed. However, Algorithm 2 may not achieve the competitive ratio of $O(\log n / \log \log n)$ as the proof does not work for this setting. Nevertheless, if we can obtain information on the execution time of completed jobs, we can reestablish an $O(\log n / \log \log n)$ -competitive algorithm by processing the jobs that are completed once but need to be reprocessed between phases.

References

- 1 Ali Allahverdi, Chi To Ng, TC Edwin Cheng, and Mikhail Y Kovalyov. A survey of scheduling problems with setup times or costs. *European journal of operational research*, 187(3):985–1032, 2008. doi:10.1016/j.ejor.2006.06.060.
- 2 Norbert Ascheuer, Sven O. Krumke, and Jörg Rambau. Online Dial-a-Ride Problems: Minimizing the Completion Time. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science*, volume 1770, pages 639–650, 2000. doi:10.1007/3-540-46541-3_53.
- 3 Evripidis Bampis, Alexander V. Kononov, Giorgio Lucarelli, and Fanny Pascual. Non-clairvoyant makespan minimization scheduling with predictions. In Satoru Iwata and Naonori Kakimura, editors, *Proceedings of the International Symposium on Algorithms and Computation*, volume 283 of *LIPIcs*, pages 9:1–9:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ISAAC.2023.9.
- 4 Alexander Birx. *Competitive analysis of the online dial-a-ride problem*. PhD thesis, Technische Universität Darmstadt, 2020.
- 5 Alexander Birx and Yann Disser. Tight analysis of the smartstart algorithm for online dial-a-ride on the line. *SIAM Journal on Discrete Mathematics*, 34(2):1409–1443, 2020. doi:10.1137/19M1268513.
- 6 Alexander Birx, Yann Disser, and Kevin Schewior. Improved bounds for open online dial-a-ride on the line. In *Proceedings of Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 145, 2019. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.21.
- 7 Vincenzo Bonifaci, Maarten Lipmann, and Leen Stougie. *Online multi-server dial-a-ride problems*. SPOR-Report : reports in statistics, probability and operations research. Technische Universiteit Eindhoven, 2006.
- 8 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. *Operations Research Forum*, 3(1):20, 2022. doi:10.1007/s43069-021-00101-z.
- 9 Srikrishnan Divakaran and Michael Saks. An online scheduling problem with job set-ups. *DIMACS Technical Report: 2000*, 34, 2000.
- 10 Konstantinos Dogeas, Thomas Erlebach, and Ya-Chun Liang. Scheduling with Obligatory Tests. In *Proceedings of the Annual European Symposium on Algorithms*, pages 48:1–48:14, 2024. doi:10.4230/LIPIcs.ESA.2024.48.
- 11 Esteban Feuerstein and Leen Stougie. On-line single-server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001. doi:10.1016/S0304-3975(00)00261-9.
- 12 Greg N. Frederickson, Matthew S. Hecht, and Chul E. Kim. Approximation Algorithms for Some Routing Problems. *SIAM Journal on Computing*, 7(2):178–193, 1978. doi:10.1137/0207017.
- 13 Giorgio Gambosi and Gaia Nicosia. On-line scheduling with setup costs. *Information Processing Letters*, 73(1–2):61–68, 2000. doi:10.1016/S0020-0190(99)00152-0.
- 14 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, volume 174. W. H. Freeman, 1979.
- 15 Hiromichi Goko, Akitoshi Kawamura, Yasushi Kawase, Kazuhisa Makino, and Hanna Sumita. Online scheduling on identical machines with a metric state space. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science*, pages 32:1–32:21, 2022. doi:10.4230/LIPIcs.STACS.2022.32.

- 16 Ronald L. Graham. Bounds for Certain Multiprocessing Anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966. doi:10.1002/j.1538-7305.1966.tb01709.x.
- 17 Dan Gusfield. Bounds for naive multiple machine scheduling with release times and deadlines. *Journal of Algorithms*, 5(1):1–6, 1984. doi:10.1016/0196-6774(84)90035-X.
- 18 Leslie A. Hall and David B. Shmoys. Approximation schemes for constrained scheduling problems. In *Proceedings of Annual Symposium on Foundations of Computer Science*, pages 134–139, 1989. doi:10.1109/SFCS.1989.63468.
- 19 Dorit S Hochbaum and David B Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987. doi:10.1145/7531.7535.
- 20 Hongtao Hu, K. K.H. Ng, and Yichen Qin. Robust Parallel Machine Scheduling Problem with Uncertainties and Sequence-Dependent Setup Time. *Scientific Programming*, 2016. doi:10.1155/2016/5127253.
- 21 Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. *ACM Transactions on Parallel Computing*, 10(4):1–26, 2023. doi:10.1145/3593969.
- 22 Sven O. Krumke. *Online optimization: Competitive analysis and beyond*. PhD thesis, Technische Universität Berlin, 2001.
- 23 Maarten Lipmann, Xiwen Lu, Willem E. de Paepe, Rene A. Sitters, and Leen Stougie. On-Line Dial-a-Ride Problems Under a Restricted Information Model. *Algorithmica*, 40(4):319–329, 2004. doi:10.1007/s00453-004-1116-z.
- 24 Alexander Mäcker, Manuel Malatyali, Friedhelm Meyer auf der Heide, and Sören Riechers. Non-clairvoyant scheduling to minimize max flow time on a machine with setup times. In *Proceedings of the International Workshop on Approximation and Online Algorithms*, pages 207–222. Springer, 2017. doi:10.1007/978-3-319-89441-6_16.
- 25 David M Miller, Hui-Chuan Chen, Jessica Matson, and Qiang Liu. A hybrid genetic algorithm for the single machine scheduling problem. *Journal of Heuristics*, 5(4):437–454, 1999. doi:10.1023/A:1009684406579.
- 26 Rajeev Motwani, Steven Phillips, and Eric Torng. Nonclairvoyant scheduling. *Theoretical computer science*, 130(1):17–47, 1994. doi:10.1016/0304-3975(94)90151-1.
- 27 Kiyohito Nagano and Akihiro Kishimoto. Subadditive load balancing, 2019. doi:10.48550/arXiv.1908.09135.
- 28 Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 2008. doi:10.1007/978-3-031-05921-6.
- 29 Kirk Pruhs, Jiri Sgall, and Eric Torng. Online scheduling. In Joseph Y.-T. Leung, editor, *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004. doi:10.1201/9780203489802.CH15.
- 30 Vinod K Sahney. Single-server, two-machine sequencing with switching time. *Operations Research*, 20(1):24–36, 1972. doi:10.1287/opre.20.1.24.
- 31 Jiří Sgall. *On-line scheduling*, pages 196–231. Springer Berlin Heidelberg, 1998.
- 32 David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24(6):1313–1331, 1995. doi:10.1137/S0097539793248317.
- 33 Zoya Svitkina and Lisa Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. *SIAM Journal on Computing*, 40(6):1715–1737, 2011. doi:10.1137/100783352.