Scheduling with Obligatory Tests

Konstantinos Dogeas
Department of Computer Science, Durham University, UK

Thomas Erlebach
Department of Computer Science, Durham University, UK

Ya-Chun Liang
Data Science Institute, Columbia University, New York, NY, USA
Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan

Abstract

Motivated by settings such as medical treatments or aircraft maintenance, we consider a scheduling problem with jobs that consist of two operations, a test and a processing part. The time required to execute the test is known in advance while the time required to execute the processing part becomes known only upon completion of the test. We use competitive analysis to study algorithms for minimizing the sum of completion times for n given jobs on a single machine. As our main result, we prove using a novel analysis technique that the natural 1-SORT algorithm has competitive ratio at most 1.861. For the special case of uniform test times, we show that a simple threshold-based algorithm has competitive ratio at most 1.585. We also prove a lower bound that shows that no deterministic algorithm can be better than $\sqrt{2}$-competitive even in the case of uniform test times.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Competitive ratio, Online algorithm, Scheduling with testing, Sum of completion times

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.48

Related Version *Full Version*: `https://arxiv.org/abs/2406.16734` [4]

Funding *Konstantinos Dogeas*: Funded by EPSRC grant EP/S033483/2.
Thomas Erlebach: Supported by EPSRC grants EP/S033483/2 and EP/T01461X/1.

1 Introduction

Settings where the processing time of a job is initially uncertain but can be determined by executing a test have received increasing attention in recent years. Levi et al. [9] considered a model where the weight and processing time of a job follow a known joint probability distribution, a job can be tested to reveal its weight and processing time, and the goal is to find a scheduling policy that minimizes the expectation of the weighted sum of completion times. Dürr et al. [5] introduced an adversarial setting of scheduling with testing where each job j is given with an upper bound u_j on its processing time. The scheduler can either execute the job untested (with processing time u_j), or test it first to reveal its actual processing time $p_j \leq u_j$ and then execute it with processing time p_j. They studied the setting of uniform test times and gave competitive algorithms for minimizing the sum of completion times and for makespan minimization on a single machine. Later work considered this model with arbitrary test times for minimizing the sum of completion times on a single machine [1, 10] or multiple machines [6], for makespan minimization on parallel machines [2, 8, 7], and for minimizing energy or maximum speed in scheduling with speed scaling [3].

In all these studies, it is *optional* for the scheduler whether to test a job or not. In many application settings, however, it is natural to assume that a test *must* be executed for each job before the job can be executed. For example, for a repair job, it is necessary to first

diagnose the fault (this corresponds to a test) before the repair can be carried out, and the result of the fault diagnosis yields information about how long the repair job will take. For a maintenance job (for example, aircraft maintenance [9]), it is necessary to determine the maintenance needs (this corresponds to a test) before the maintenance can be carried out. In a medical emergency department, patients need to be diagnosed (i.e., "tested") before they can be treated. Therefore, we propose to study scheduling with testing in a setting with *obligatory tests*. Initially, each job j is given with a test time t_j, and nothing is known about its processing time. Testing the job takes time t_j and reveals the processing time p_j of the job. The processing part of the job can then be scheduled any time after the completion of the test and takes time p_j to be completed. We study algorithms for minimizing the sum of the completion times on a single machine and evaluate the performance of our algorithms using competitive analysis.

For minimizing the sum of completion times on a single machine in our setting with obligatory tests, obtaining a 2-competitive algorithm is straightforward: Treating each job (test plus processing part) as a single entity with unknown processing time and applying the Round Robin (RR) algorithm (which executes all unfinished jobs simultaneously at the same rate) gives a 2-competitive preemptive schedule [11], and in our setting this algorithm can be made non-preemptive without any increase in job completion times: At any time, among all tests or processing parts currently available for execution, it is known which of them will complete first in the preemptive schedule, and hence that test or processing part can be chosen to be executed non-preemptively first (the same observation has been made previously for the setting with optional tests [5, 10, 6]). Our aim is therefore to design algorithms that are better than 2-competitive.

Our contributions. For the setting with arbitrary test times, we consider the algorithm 1-SORT, which is a natural adaptation of the (α, β)-SORT algorithm proposed by Albers and Eckl [1] to the setting with obligatory tests. Using a novel analysis technique that we consider our main contribution, we show that the competitive ratio of 1-SORT is at most 1.861. In our analysis, we consider a complete graph on the jobs, where each edge is associated with the delay that the two jobs connected by the edge create for each other. The sum of the delays associated with the edges and the job processing times is then equal to the sum of completion times of the schedule. The graph can contain edges where the associated delay in the schedule computed by the algorithm is arbitrarily close to twice the delay in the optimal schedule, and therefore a straightforward analysis would only yield a competitive ratio of 2. We show that for edges with delay ratio close to 2 there are always sufficiently many other edges whose delay ratio is much smaller than 2, so that overall the ratio of the objective values of the algorithm and the offline optimum is bounded by a value smaller than 2.

For the setting with unit test times, we consider an adaptation of the Threshold algorithm by Dürr et al. [5] to the setting with obligatory tests: When the test of a job reveals a processing time smaller than a threshold y, the algorithm executes the processing part of the job immediately; otherwise, the execution of the processing part is deferred to the end of the schedule, where all the processing parts that have been deferred are executed in SPT (shortest processing time) order. We show that the algorithm is 1.585-competitive (and this analysis is tight for the algorithm). We also give a lower bound showing that no deterministic algorithm can be better than $\sqrt{2}$-competitive.

The remainder of the paper is structured as follows. After discussing related work in the remainder of this section, we give a formal problem definition and discuss preliminaries in Section 2. Our algorithm for arbitrary test times and its analysis are presented in Section 3.

The threshold-based algorithm and its analysis as well as the lower bound for uniform test times are given in Section 4. Conclusions are presented in Section 5. Statements where the proofs have been omitted due to space restrictions are marked by ($\star$); the proofs of these statements can be found in the full version of the paper [4].

Related work. For the classical offline scheduling problem (without tests) of minimizing the sum of completion times on a single machine, denoted by $1 \,||\, \sum C_j$, it is known that always executing first the job with the shortest processing time (SPT) among all unscheduled jobs gives the optimal schedule (a generalisation to the weighted sum of completion times was proven by Smith [12]). For the setting with unknown processing times (i.e., the scheduler does not know the processing time of a job until the job completes), Motwani et al. [11] showed that the Round Robin (RR) algorithm, a preemptive algorithm that schedules all unfinished jobs simultaneously, is $\left(2 - \frac{2}{n+1}\right)$-competitive, where n is the number of jobs, and that this is best possible.

As mentioned earlier, Dürr et al. [5] introduced the adversarial model for scheduling with testing in a setting with optional tests: For each job j its test time t_j and an upper bound u_j on its processing time are given. The algorithm can either execute the job untested with processing time u_j or test it first. The test takes time t_j and reveals the actual processing time p_j, which satisfies $0 \leq p_j \leq u_j$. The job can then be executed at any time after the test and takes time p_j. They considered only the case of uniform test times ($t_j = 1$ for all jobs j) and provided a 2-competitive deterministic algorithm and a 1.7453-competitive randomized algorithm for minimizing the sum of completion times on a single machine. Their deterministic 2-competitive algorithm is the algorithm Threshold that tests all jobs with $u_j \geq 2$ and executes the processing part of a job j immediately after its test if $p_j \leq 2$ and otherwise defers the job to the end of the schedule (where the processing parts of all unfinished jobs are executed in SPT order). They also gave lower bounds of 1.8546 and 1.6257 for deterministic and randomized algorithms, respectively. Albers and Eckl [1] considered the problem with arbitrary test times and gave a deterministic 4-competitive algorithm, a 3.3794-competitive randomized algorithm, and a preemptive deterministic algorithm with competitive ratio $2\phi \approx 3.2361$, where $\phi = (1 + \sqrt{5})/2$ is the golden ratio. Their preemptive deterministic algorithm can be made non-deterministic as outlined above, thus giving a 2ϕ-competitive deterministic algorithm. The algorithm for which they showed competitive ratio 4 is called (α, β)-SORT: It tests a job j if $u_j \geq \alpha t_j$ and, at any time, executes the test or processing part of smallest priority, where the priority of the test of a job j is taken to be βt_j and the priority of the processing part of a tested job j is taken to be p_j. In their analysis, choosing $\alpha = \beta = 1$ optimizes the resulting ratio, giving the bound of 4. Liu et al. [10] showed that a more careful analysis of (α, β)-SORT yields that the algorithm achieves ratio $1 + \sqrt{2} \approx 2.414$ for $\alpha = \beta = \sqrt{2}$. They also gave improved algorithms with deterministic competitive ratio 2.316513 and randomized competitive ratio 2.152271. Gong et al. [6] considered the problem of minimizing the sum of completion times in the setting with optional tests on multiple machines. Among other results, they presented a 3.2361-competitive algorithm for arbitrary test times and an algorithm with competitive ratio approaching 2.9271 for large m for uniform test times.

2 Problem definition and preliminaries

Problem definition. We are given a job set $\mathcal{J} = \{1, 2, \ldots, n\}$ to be scheduled on a single machine. Each job $j \in \mathcal{J}$ has an *unknown* processing time $p_j \geq 0$ and a *known* test time $t_j \geq 0$, where p_j and t_j are non-negative real numbers. We denote the *total size* (or just

size) of job j by $\sigma_j = t_j + p_j$. Furthermore, we denote the maximum of the test time and the processing time of job j by $m_j = \max\{t_j, p_j\}$. Testing job j takes time t_j and reveals its processing time p_j. Once job j has been tested, its processing part can be executed and takes time p_j. The completion time C_j of a job is the point in time when its processing part finishes. We consider the setting with *obligatory tests* where every job must be tested before the processing part of the job can be executed. Note that the test of every job must be executed both by the algorithm and by the optimal solution. The machine can execute at any time only one test or one processing part of a job. The tests and processing parts must be scheduled non-preemptively, but the processing part of a job does not have to be started immediately after its test. As is common in the literature on scheduling with testing for minimizing the sum of completion times [5, 1, 6], we refer to this setting as *non-preemptive* but note that it has been called *test-preemptive* in the context of makespan minimization [2, 8, 7]. The objective is to minimize the sum of completion times $\sum_{j\in\mathcal{J}} C_j$.

In the setting of *uniform test times*, we assume that $t_j = 1$ for all $j \in \mathcal{J}$. In the setting of *arbitrary test times*, the test time of each job j is an arbitrary real number $t_j \geq 0$. Using Graham's notation for describing scheduling problems, these two variations can be denoted by $1 \mid t_j = 1 \mid \sum_j C_j$ and $1 \mid t_j \mid \sum_j C_j$, respectively.

The objective function. For the purpose of analyzing the competitive ratio of algorithms, it will be useful to consider different ways of expressing the objective function. For two different jobs k and j in the schedule produced by the algorithm under consideration, we use $d_{k,j}$ to denote the amount of time that the test and/or processing part of job k get executed before the completion of job j. The completion time of job j can then be written as $C_j = \sigma_j + \sum_{k\in\mathcal{J},k\neq j} d_{k,j}$. For any pair of different jobs j and k, we use $D(j,k) = d_{j,k} + d_{k,j}$ to denote the delay that job j causes for job k plus the delay that job k causes for job j. Thus:

$$\sum_{j\in\mathcal{J}} C_j = \sum_{j\in\mathcal{J}} (\sigma_j + \sum_{\substack{k\in\mathcal{J}\\k\neq j}} d_{k,j}) = \sum_{j\in\mathcal{J}} \sigma_j + \sum_{\substack{j,k\in\mathcal{J}\\j<k}} D(j,k) \tag{1}$$

The optimal schedule. An optimal offline schedule views each job as a single operation that takes total time σ_j to be executed and schedules the jobs in SPT order with respect to those times. We use $d^*_{j,k}$ and $D^*(j,k)$ to denote the values corresponding to $d_{j,k}$ and $D(j,k)$ in the optimal schedule. For jobs k and j with $\sigma_k < \sigma_j$, we have $d^*_{k,j} = \sigma_k$, $d^*_{j,k} = 0$ and $D^*(j,k) = \sigma_k$. In general, $D^*(j,j') = \min\{\sigma_j, \sigma_{j'}\}$ for any pair of jobs j and j'. For the sum of completion times OPT in the optimal schedule, we have:

$$OPT = \sum_{j\in\mathcal{J}} \sigma_j + \sum_{\substack{j,k\in\mathcal{J}\\j<k}} D^*(j,k) = \sum_{j\in\mathcal{J}} \sigma_j + \sum_{\substack{j,k\in\mathcal{J}\\j<k}} \min\{\sigma_j, \sigma_k\} \tag{2}$$

Competitive ratio. For an algorithm under consideration, we use ALG to denote the sum of completion times in the schedule produced by the algorithm for a given instance. By OPT we denote the sum of completion times in the optimal offline schedule for that instance. We say that the algorithm is ρ-*competitive* (or has *competitive ratio* at most ρ) if $ALG/OPT \leq \rho$ holds for all instances of the problem.

3 Arbitrary test times

In this section, we consider the problem $1 \mid t_j \mid \sum C_j$ where the test times can be arbitrary non-negative real numbers. We refer to the test and the processing part of a job j as *operations* and denote the test operation by τ_j and the processing operation by π_j. First, we present the algorithm β-SORT. Then, we prove an upper bound of 1.861 on the competitive ratio of 1-SORT. Finally, we present input examples showing that the competitive ratio of β-SORT is at least 1.618 for $\beta = 1$ and at least some larger value for all other values of β.

Algorithm 1 β-SORT.

```
1 R = ∅; // empty priority queue
2 for j ∈ J do
3     insert the test operation τ_j with priority β × t_j into R
4 while R ≠ ∅ do
5     o = R.deleteMin();
6     execute o;
7     if o was the test operation τ_j of a job j then
8         insert the processing operation π_j with priority p_j into R
```

Algorithm β-SORT. For the problem variant with optional tests, Albers and Eckl [1] proposed the algorithm (α, β)-SORT that tests a job j if $u_j \geq \alpha t_j$ and always schedules the shortest available operation, but uses $\beta \times t_j$ instead of t_j when comparing the test time of job j with the processing time of another job that has already been tested. They showed that the algorithm is 4-competitive with $\alpha = \beta = 1$. We adapt their algorithm to our setting with obligatory tests. The parameter α is not relevant in our setting as every job must be tested, so we refer to the resulting algorithm as β-SORT (see Algorithm 1). The algorithm maintains a priority queue $\mathcal{R}$ of available test and processing operations (i.e., the test operations of jobs that have not yet been tested and the processing parts of jobs that have already been tested). The priority of a test operation τ_j is $\beta \times t_j$ and the priority of a processing operation π_j is p_j. The algorithm always schedules next the operation with minimum priority in $\mathcal{R}$ (returned and removed from $\mathcal{R}$ by the call to $\mathcal{R}$.deleteMin()) and, if that operation was a test, inserts the corresponding processing operation into $\mathcal{R}$.

Upper bound on the competitive ratio of 1-SORT. By adapting the analysis by Albers and Eckl [1] in a straightforward way, one gets that β-SORT is $\left(1 + \max\left\{1 + \frac{1}{\beta}, 1 + \beta\right\}\right)$-competitive. This bound is minimized for $\beta = 1$, showing that the competitive ratio of 1-SORT is at most 3. We fix $\beta = 1$ and prove the substantially better bound of 1.861 on the competitive ratio of 1-SORT. We do not believe that β-SORT with a value of β different from 1 has a better competitive ratio than that obtained with $\beta = 1$ in our setting; adapting our analysis to values of β different from 1, we found that the resulting bound on the competitive ratio became larger.

First, we give an intuitive overview of our analysis. We consider an oriented complete graph $G = (V, A)$ where $V = \mathcal{J}$ and each edge is directed towards the job with larger size. We write jk for the arc (directed edge) from j to k. By (1), we can view the sum of completion times of a schedule as if it was produced by a contribution σ_j of each vertex $j \in V$ and a contribution $D(j, k)$ of each arc $jk \in A$. The contributions of the vertices are the same in

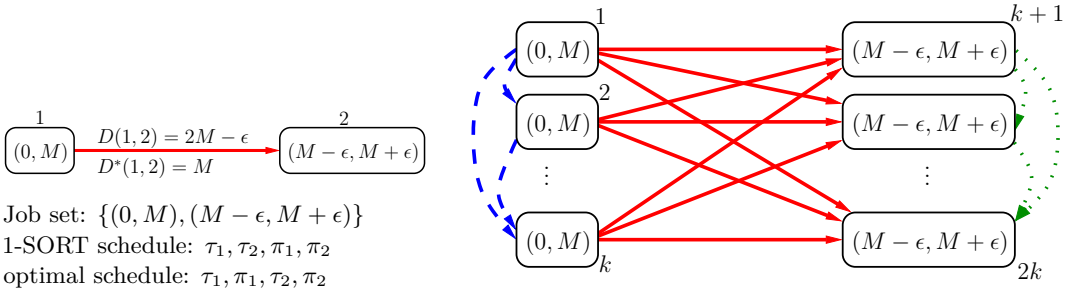

Figure 1 Left: Instance with two jobs where the delay ratio on the arc $(1, 2)$ is arbitrarily close to 2. Right: Instance with $2k$ jobs to illustrate red (drawn solid), blue (drawn dashed) and green (drawn dotted) arcs. A job with test time t_j and processing time p_j is written as a pair (t_j, p_j).

the algorithm's schedule and in the optimal schedule. If the *delay ratio* $D(j, k)/D^*(j, k)$ is bounded by ρ for every arc jk, it follows that $ALG/OPT \leq \rho$. Unfortunately, the delay ratio of an individual arc can be arbitrarily close to 2. Consider for example an instance with two jobs with $t_1 = 0$, $p_1 = M$ and $t_2 = M - \epsilon$, $p_2 = M + \epsilon$ for a large constant M and an infinitesimally small $\epsilon > 0$ (see Fig. 1 (left)). Algorithm 1-SORT schedules the operations in the order $\tau_1, \tau_2, \pi_1, \pi_2$ giving $D(1, 2) = t_1 + p_1 + t_2 = 2M - \epsilon$, while the optimal schedule is $\tau_1, \pi_1, \tau_2, \pi_2$ with $D^*(1, 2) = t_1 + p_1 = M$. Hence, the delay ratio on the arc $(1, 2)$ is arbitrarily close to 2. Nevertheless, the ratio ALG/OPT on this example does not exceed $5/4$, as the term $\sigma_1 + \sigma_2 = 3M$ that makes the same contribution to ALG and OPT is relatively large compared to the delays on the arc $(1, 2)$. We refer to arcs with large delay ratios (to be defined precisely later on) as *red* arcs. The example suggests the idea of analyzing red arcs together with other terms contributing to the objective function in order to show a competitive ratio smaller than 2.

In the example of Fig. 1 (left) it was enough to consider the red arc $(1, 2)$ together with the contributions to the objective value made by vertices 1 and 2, but this kind of argument cannot suffice in general because the number of arcs is quadratic in the number of vertices. Consider the example of a job set with $n = 2k$ jobs that contains k copies of each of the jobs from the previous example (see Fig. 1 (right)). We call the k jobs with $t_j = 0$, $p_j = M$ *left* jobs and the k jobs with $t_j = M - \epsilon$, $p_j = M + \epsilon$ *right* jobs in the following. There are now k^2 arcs between left and right jobs, each with a delay ratio arbitrarily close to 2. The contribution $k \cdot M + k \cdot 2M$ that the $2k$ vertices make to the objective function is no longer sufficient to show a bound smaller than 2 for the competitive ratio, as it is negligible (for large k) compared to the total delay on all the k^2 arcs between left and right jobs, which is $k^2(2M - \epsilon)$ for 1-SORT and $k^2 M$ in the optimal schedule. What we can exploit here instead is that the $k(k-1)/2$ arcs between left jobs have the same delay M in the schedule produced by 1-SORT and in the optimal schedule (delay ratio 1), and that the $k(k-1)/2$ arcs between right jobs have delay $2M$ in the optimal schedule and delay $3M - \epsilon$ in the schedule produced by 1-SORT (delay ratio ≈ 1.5). We refer to the arcs between left jobs as *blue* arcs and to the arcs between right jobs as *green* arcs. The total delay on all the blue, red and green arcs in this example is approximately $\frac{k^2}{2}M + k^2 \cdot 2M + \frac{k^2}{2} \cdot 3M = 4k^2 M$ for 1-SORT and

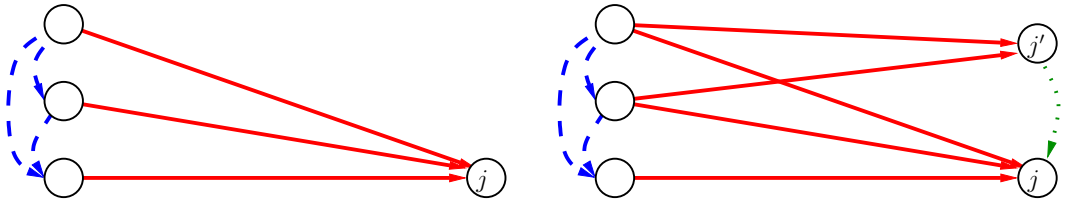

Figure 2 Illustration of the idea underlying the analysis of a red vertex j: If the blue arcs (drawn dashed) have not yet been used in the analysis of a previous red vertex, they can be used in combination with the incoming red arcs (drawn solid) of j (left). If blue arcs have already been used in the analysis of a previous red vertex j', there must be a green arc (drawn dotted) between j' and j that is also available to be used in the analysis of the incoming red arcs of j.

approximately $\frac{k^2}{2}M + k^2M + \frac{k^2}{2} \cdot 2M = 2.5k^2M$ for the optimal schedule, where we have set $\epsilon = 0$ and omitted terms linear in k. Thus, analyzing the red arcs together with the green and blue arcs is sufficient to show that $ALG/OPT \leq 4/2.5 = 1.6$ in this example.

To turn these observations into a rigorous analysis, we will proceed as follows: We give a formal definition of red arcs and refer to the vertices with incoming red arcs as red vertices. We then consider the red vertices in order of increasing t_j. For a red vertex j, we would like to analyze the delay of its incoming red arcs together with the blue arcs between their tail vertices. If those blue arcs have not been used in the analysis of previously considered red vertices, that suffices. If some of those blue arcs have already been used in the analysis of previously considered red vertices, however, then we can additionally use the green arcs that those previously considered red vertices have to j in order to make up for the unavailability of blue arcs. The crux of the analysis is a carefully specified invariant that ensures that there are always sufficiently many green arcs available for the analysis of a red vertex to make up for blue arcs that have been used in the analysis of previously considered red vertices. See Fig. 2 for an illustration of this idea. Overall, the outcome is that the incoming red arcs of each red vertex can be analyzed together with a sufficient number of blue and green arcs (which are not used for the analysis of any other red vertex) to get a ratio smaller than 2. One slight complication is that an arc may play the role of a green arc for one red vertex and the role of a blue arc for another red vertex, but we can handle this by treating that arc as a combination of a distinct *special* blue arc and a distinct *special* green arc.

Having given an intuitive overview of the ideas underlying our analysis of 1-SORT, we now proceed to present the technical details. We use two parameters $\mu > 1$ and ν with $0 < \nu < 1$, satisfying $\mu > 1/\nu$ and $1 + \frac{1}{\mu} \leq \nu + \nu^2$. Intuitively, the parameter μ determines which jobs we view as imbalanced (having a large factor between test time and processing time), and the parameter ν determines when we view a test time or processing time to be "not much smaller" than another value (namely, when it is at least ν times the other value).

▶ **Theorem 1.** *The β-SORT algorithm with $\beta = 1$ has competitive ratio at most ρ with*

$$\rho = \max\left\{ \frac{\nu+\nu^2+2+\frac{2}{\mu}}{\nu+\nu^2+1+\frac{1}{\mu}}, 1+\frac{1}{2+\nu}, \frac{\frac{4}{\nu}+\frac{4}{\mu\nu}+\nu+\nu^2+1}{\frac{2}{\nu}+\frac{2}{\mu\nu}+\nu+\nu^2}, \frac{\frac{4}{\nu}+\frac{4}{\nu\mu}+\nu+\frac{1}{\mu+1}}{\frac{2}{\nu}+\frac{2}{\mu\nu}+\nu}, \right.$$
$$\left. \frac{4+\frac{5}{\mu}+\nu}{2+\frac{2}{\mu}+\nu}, 1+\frac{1}{\nu+\nu^2}, 1+\frac{1}{\nu(\mu+1)}, \frac{2\mu+1}{\mu+1}, 1+\nu \right\}$$

We will prove Theorem 1 in the remainder of this section. Choosing μ and ν so as to minimize the ratio of Theorem 1 (done using Mathematica) yields the following:

▶ **Corollary 2.** *The ratio ρ of Theorem 1 is minimized for $\mu = \mu_0 \approx 6.16277$ and $\nu = \nu_0 \approx 0.860389$, yielding that β-SORT with $\beta = 1$ has competitive ratio at most 1.86039. Here μ_0 is the only real root of the polynomial $-2 - 8\mu - 13\mu^2 - 11\mu^3 - 4\mu^4 + \mu^5$ and $\nu_0 = \frac{\mu_0}{\mu_0+1}$. The ratio is $\rho = \frac{1+2\mu_0}{1+\mu_0}$.*

As discussed in Section 2, the optimal schedule executes the jobs in SPT order (with respect to their size), giving the objective value stated in Equation (2).

Using infinitesimal perturbations of the test times and processing times of the jobs that do not affect the schedule produced by 1-SORT nor the optimal schedule, we can assume without loss of generality that no two values in the set of the test times, processing times, and sizes of all jobs are equal. Therefore, when we compare any two such values, we can always assume that strict inequality holds.

For the purpose of the analysis, we create an auxiliary graph $G = (V, A)$, with $|V| = n$ and $|A| = \binom{n}{2} = \frac{n(n-1)}{2}$. Each vertex represents a job (both the testing and processing operation), and there is a single arc between any two vertices. The arc between vertices j and k is directed towards k if $\sigma_j < \sigma_k$ and towards j otherwise. Recall that we write jk for an arc directed from j to k. In addition, we associate with each arc jk the values $D(j,k)$ and $D^*(j,k)$ that represent the pairwise delay between jobs j and k in the schedule produced by 1-SORT and in the optimal schedule, respectively, and the *delay ratio* $\rho_{jk} = D(j,k)/D^*(j,k)$.

By (1) and (2), we have $ALG = \sum_{j\in V} \sigma_j + \sum_{jk\in A} D(j,k)$ and $OPT = \sum_{j\in V} \sigma_j + \sum_{jk\in A} D^*(j,k)$. Note that the first sum is the same in both expressions and therefore contributes to ALG and OPT in the same way, while the second sum, which represents the pairwise delays among all jobs, differs. As discussed earlier, the difficulty when aiming to show competitive ratio smaller than 2 is that there may exist arcs jk for which $D(j,k)$ can be arbitrarily close to $2 \cdot D^*(j,k)$. Hence, we cannot hope to prove a bound better than $\rho_{jk} \leq 2$ for all arcs jk, and such a bound would only yield $ALG/OPT \leq 2$. As each job j contributes σ_j to both ALG and OPT, we say that the *delay ratio* of job j, denoted by ρ_j, is equal to 1. In order to prove a competitive ratio better than 2, we need to show that arcs jk with delay ratio close to 2 can be analysed together with arcs for which the delay ratio is much smaller than 2 and/or together with vertices, for which we know that the delay ratio is 1. This then yields that the ratios ρ_{jk} and ρ_j are bounded by a constant smaller than 2 *on average*.

It turns out that the ratio of an arc jk can be close to 2 only if job j is *imbalanced* and the test time of job k is smaller but not much smaller than $m_j = \max\{t_j, p_j\}$, and p_k is not much smaller than t_k.

▶ **Definition 3.** *A job j is called* imbalanced *if $m_j = \max\{t_j, p_j\} \geq \mu \cdot \min\{t_j, p_j\}$, for a fixed constant $\mu > 1$.*

▶ **Definition 4.** *An arc jk is called a* red *arc if j is imbalanced, $m_j \geq t_k \geq \nu \cdot m_j$, and $p_k \geq \nu \cdot t_k$. Here, ν is a constant with $0 < \nu < 1$ that satisfies $\mu > \frac{1}{\nu}$ and $1 + \frac{1}{\mu} \leq \nu + \nu^2$.*

▶ **Lemma 5** (⋆). *If jk is a red arc, then $\sigma_j \leq \sigma_k$.*

The following lemma can be proved by considering each leaf of a decision tree for determining the order in which 1-SORT executes the four operations of the two jobs j and k, and showing that $D(j,k)/D^*(j,k)$ is bounded by the claimed upper bound if the arc jk does not satisfy at least one of the three conditions of Definition 4.

▶ **Lemma 6** (⋆). *If an arc jk is not red, then $\rho_{jk} \leq \max\{(2\mu+1)/(\mu+1), 1+\nu\}$.*

By considering the same decision tree, it is also easy to see that $D(j,k) \leq 2D^*(j,k)$ holds for all arcs (including red arcs). In the following, we will show that, for each job with incoming red arcs, those arcs can be grouped together with a set of non-red arcs and the size of the job in such a way that the total ratio of the algorithm's delay over the optimal delay for the group is bounded by a constant ρ that is smaller than 2.

Let V_I be the set of all imbalanced jobs, ordered by non-decreasing m_j. Let V_R be the set of jobs with at least one incoming red arc. (If V_R is empty, the competitive ratio of the algorithm is bounded by the ratio of Lemma 6.) Consider the jobs in V_R to be sorted in order of non-decreasing test times and write $i \prec j$ if the test time of i comes before the test time of j in that order. Consider a particular job $k \in V_R$ with test time t_k. Every incoming red arc jk of k must come from a job j that is imbalanced and satisfies $m_j \geq t_k \geq \nu \cdot m_j$ and, as j is imbalanced, $\min\{t_j, p_j\} \leq m_j/\mu$. Let $N^-(k)$ be the set of vertices that have an outgoing red arc to k, i.e., $N^-(k) = \{j \mid jk \text{ is a red arc}\}$. For a vertex $j \in N^-(k)$, let $N^-_{\geq j}(k)$ be the subset of $N^-(k)$ that consists of j and all vertices of $N^-(k)$ that come after j (in the order of V_I). Furthermore, let $P(k)$ denote the set of jobs coming before k in V_R (in the order $\prec$) that also have an incoming red arc from at least one job in $N^-(k)$.

We process the jobs in V_R in $\prec$-order. To handle a job $k \in V_R$, we consider the subgraph G_k of G induced by $V_k = \{k\} \cup N^-(k) \cup P(k)$. We call arcs between two jobs in $N^-(k)$ *blue* and arcs between k and any job in $P(k)$ *green*. The directions of blue and green arcs are irrelevant and can be ignored. We denote by C_k the set of elements (vertices and arcs) of G that are grouped with the red incoming arcs of k for the analysis. We will always have that k and its incoming red arcs are in C_k, and we will add a suitable number of blue and/or green arcs to C_k. Each blue and/or green arc will be added to at most one such set C_k, except in a special case where an arc e plays the role of a blue arc for one k and the role of a green arc for another k; in that case, we will split e into a green arc and a blue arc, and each part will be added to at most one set C_k. We let ρ_{C_k} denote the ratio of the sum of the delays on all the arcs and vertices in C_k in the solution by the algorithm divided by the sum of the delays on the same arcs and vertices in the optimal schedule.

▶ **Lemma 7** ($\star$). *For any $k \in V_R$, the set $N^-(k)$ is a contiguous subset of V_I.*

▶ **Lemma 8** ($\star$). *Let job k be a job with incoming red arcs, and let $r \in P(k)$. Then the intersection of $N^-(k)$ and $N^-(r)$ is a (not necessarily proper) prefix of $N^-(k)$. Furthermore, $N^-(r)$ cannot contain any vertex in V_I that comes after $N^-(k)$.*

We say that a blue arc is *used* or *used up* in the analysis of a vertex $k \in V_R$ if the arc is added to the set C_k. We maintain the following invariant when processing the vertices in V_R.

▶ **Invariant 1.** *Consider a vertex $k \in V_R$, and any vertex j in $N^-(k)$. Let $P_{\geq j}(k)$ be the set of vertices in V_R that have been processed before k and that have a red arc from j. For each $r \in \{k\} \cup P_{\geq j}(k)$, let $o_{\geq j}(r) = |N^-_{\geq j}(r)|$. Then the total number of blue arcs between vertices in $N^-_{\geq j}(k)$ that have been used up in the analysis of vertices in $\{k\} \cup P_{\geq j}(k)$ at the time when k has just been processed is at most $\sum_{r \in \{k\} \cup P_{\geq j}(k)} (o_{\geq j}(r) - 1)$.*

Intuitively, if we imagine the vertices of V_I arranged from top to bottom in order of increasing m_j, the invariant says that for any vertex j in V_I the following condition holds: The number of blue arcs between vertices below j (including j) that have been used up in the analysis of vertices in V_R that have already been processed is bounded by the sum, over all those vertices, of their number of incoming red arcs from vertices below j minus one. See Fig. 3 for an illustration. In that figure, vertex r has $o_{\geq j}(r) = 1$ incoming red arc from $N^-_{\geq j}(k)$, s has $o_{\geq j}(s) = 2$, and k has $o_{\geq j}(k) = 3$. Therefore, the invariant says that, after k

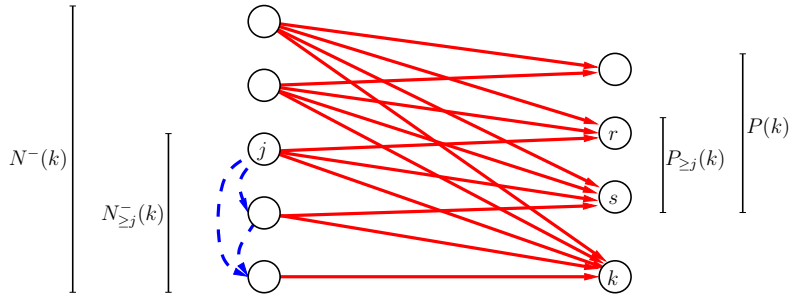

Figure 3 Illustration of Invariant 1. Only blue (dashed) arcs between vertices in $N^-_{\geq j}(k)$ are shown.

has been processed, the number of blue arcs between vertices in $N^-_{\geq j}(k)$ that have been used up is at most $(1-1)+(2-1)+(3-1)=3$. Note that Invariant 1 trivially holds before any vertices in V_R are processed because no blue arcs have been used at that point.

Next, we establish some properties of blue and green arcs.

▶ **Lemma 9** (⋆). *For each blue arc ij in G_k, we have $D^*(i,j) \geq t_k$ and $\rho_{ij} \leq \rho^B = 1 + \frac{1}{\mu\nu}$.*

▶ **Lemma 10** (⋆). *For each green arc jk (with $j \in P(k)$) in G_k, we have $D^*(j,k) \geq (\nu+\nu^2)t_k$ and $\rho_{jk} \leq \rho^G = 1 + \frac{1}{\nu+\nu^2}$.*

Unfortunately, it is possible that an arc ij is used as a blue arc in the analysis of one vertex r in V_R and as a green arc in the analysis of another vertex k in V_R. We handle this case by splitting such an arc ij into two arcs for the purpose of the analysis, a *special blue arc* used in the analysis of r and a *special green arc* used in the analysis of k. In this way, we can ensure that every arc is used in the analysis of at most one vertex. We refer to the full version for details and state here only the properties of the resulting special arcs.

▶ **Lemma 11** (⋆). *For each special blue arc $i_b j_b$ in G_k, we have $D^*(i_b, j_b) \geq t_k$ and $\rho_{i_b j_b} \leq \rho^S = 1 + \frac{1}{\nu(\mu+1)}$.*

▶ **Lemma 12** (⋆). *For each special green arc $r_g k_g$ (with $r \in P(k)$) in G_k, we have $D^*(r_g, k_g) \geq \nu t_k$ and $\rho_{r_g k_g} \leq \rho^S = 1 + \frac{1}{\nu(\mu+1)}$.*

The following lemma deals with vertices in V_R that have a single incoming red arc. It shows that no blue arcs need to be used for such vertices, so they do not play any role in the process of maintaining Invariant 1.

▶ **Lemma 13** (⋆). *If $|N^-(k)| = 1$ and we take $C_k = \{k, jk\}$, where jk is the single incoming red arc of k, then the ratio ρ_{C_k} of the algorithms's delay over the optimal delay in C_k is bounded by $\frac{\nu+\nu^2+2+\frac{2}{\mu}}{\nu+\nu^2+1+\frac{1}{\mu}}$.*

The following is the key lemma that shows for each vertex k in V_R that, assuming Invariant 1 holds before vertex k is processed, we can construct a set C_k that allow us to charge the red incoming arcs while maintaining Invariant 1.

▶ **Lemma 14** (⋆). *Let k be a vertex in V_R and assume that Invariant 1 holds just before k is processed. We can define a set C_k consisting of blue arcs connecting vertices in $N^-(k)$, green arcs connecting k with vertices in $P(k)$, all incoming red arcs of k, and k itself in such*

a way that $\rho_{C_k} \leq \rho^C$ *with*

$$\rho^C = \max\{\frac{\nu+\nu^2+2+\frac{2}{\mu}}{\nu+\nu^2+1+\frac{1}{\mu}}, 1+\frac{1}{2+\nu}, \frac{\frac{4}{\nu}+\frac{4}{\mu\nu}+\nu+\nu^2+1}{\frac{2}{\nu}+\frac{2}{\mu\nu}+\nu+\nu^2}, \frac{\frac{4}{\nu}+\frac{4}{\nu\mu}+\nu+\frac{1}{\mu+1}}{\frac{2}{\nu}+\frac{2}{\mu\nu}+\nu}, \frac{4+\frac{5}{\mu}+\nu}{2+\frac{2}{\mu}+\nu}, \rho^G, \rho^S\}.$$

Furthermore, Invariant 1 still holds after k *is processed.*

By Lemma 14, we know that for every $k \in V_R$ there is a set C_k of vertices and arcs such that $\rho_{C_k} \leq \rho^C$. Furthermore, the sets C_k are pairwise disjoint; if an arc is used as a blue arc in one set and as a green arc in another set, it is split into a special blue arc and a special green arc, and each set uses one of the two special arcs. Let V' denote the vertices that are not in any C_k and note that any vertex $j \in V'$ delays itself by σ_j in both the optimal schedule and the algorithm's schedule. Let A' denote the arcs that are not in any C_k and note that any arc $ij \in A'$ has $D(i,j) \leq \rho^N D^*(i,j)$ with $\rho^N = \max\left\{\frac{2\mu+1}{\mu+1}, 1+\nu\right\}$ by Lemma 6. We use $D(C_k)$ and $D^*(C_k)$ to denote the sum of the delays in C_k in the algorithm's schedule and in the optimal schedule, respectively. As the competitive ratio is

$$\frac{ALG}{OPT} = \frac{\sum_{j\in V'}\sigma_j + \sum_{ij\in A'} D(i,j) + \sum_{k\in V_R} D(C_k)}{\sum_{j\in V'}\sigma_j + \sum_{ij\in A'} D^*(i,j) + \sum_{k\in V_R} D^*(C_k)} \leq \frac{\sum_{j\in V'}\sigma_j + \sum_{ij\in A'} \rho^N D^*(i,j) + \sum_{k\in V_R} \rho^C D^*(C_k)}{\sum_{j\in V'}\sigma_j + \sum_{ij\in A'} D^*(i,j) + \sum_{k\in V_R} D^*(C_k)},$$

we get that the ratio is bounded by $\max\{1, \rho^C, \rho^N\} = \max\{\rho^C, \rho^N\}$. This completes the proof of Theorem 1.

Lower bounds on the competitive ratio of β-SORT. For $\beta \leq 1$, consider the following instance of the problem (where M is a fixed positive number and $\epsilon > 0$ is infinitesimally small): γn *short* jobs with $t_j = 0, p_j = M$ and $(1-\gamma)n$ *long* jobs with $t_j = \frac{M}{\beta} - 2\epsilon, p_j = M - \epsilon$. The algorithm schedules the γn tests of short jobs, then the $(1-\gamma)n$ tests of long jobs, then the processing operations of the long jobs, and finally the processing operations of the short jobs. The optimal schedule schedules first all short jobs and then all long jobs. One can show that, when choosing γ (as a function of β) to maximize the ratio ALG/OPT, then that ratio approaches $\frac{1}{2}(\sqrt{\frac{\beta+4}{\beta}}+1)$ for large n.

For $\beta \geq 1$, consider the following instance of the problem (where $\epsilon > 0$ is again infinitesimally small): γn *short* jobs with $t_j = 1+2\epsilon, p_j = 0$ and $(1-\gamma)n$ *long* jobs with $t_j = 1, p_j = \beta + \epsilon$. The algorithm schedules the tests of the long jobs, then the processing parts of the long jobs, then the short jobs (with each test followed immediately by the execution of the tested job). The optimum schedule schedules first all short jobs and then all long jobs. One can show that, when choosing γ (as a function of β) to maximize the ratio ALG/OPT, then that ratio approaches $(\sqrt{4\beta(\beta^2+\beta-1)+1}+1)/2\beta$ for large n.

This shows that β-SORT with $\beta = 1$ is not better than 1.618-competitive, and for every $\beta \neq 1$ we get a larger lower bound on the competitive ratio of β-SORT.

4 Uniform test times

In this section we assume that $t_j = 1$ for all jobs j. First, we show the following lower bound.

▶ **Theorem 15.** *No deterministic algorithm can have competitive ratio strictly smaller than* $\sqrt{2}$ *for the setting with obligatory tests and uniform test times.*

Proof. Let an arbitrary deterministic algorithm for the problem be given. Consider the following adversarial construction, with a parameter γ, $0 \leq \gamma \leq 1$, whose value will be determined later: The adversary presents n jobs. For the first γn jobs that are tested by the algorithm, the adversary sets $p_j = 1$. For the remaining $(1-\gamma)n$ jobs, the adversary sets $p_j = 0$. We call the jobs with $p_j = 0$ *short* jobs and those with $p_j = 1$ *long* jobs.

The optimal schedule will schedule the jobs in SPT order, i.e., it will first execute the $(1-\gamma)n$ short jobs and after that the γn long jobs, always executing the processing part of a job right after its test. The objective value of the optimal schedule can be written as the sum of three parts:

$$P_1 = \sum_{j=1}^{(1-\gamma)n} j = \frac{(1-\gamma)n((1-\gamma)n+1)}{2} \qquad P_2 = (1-\gamma)n \cdot \gamma n = \gamma(1-\gamma)n^2$$
$$P_3 = \sum_{j=1}^{\gamma n}(2j) = \gamma n(\gamma n + 1)$$

Here, P_1 is the sum of the completion times of the short jobs, P_2 is the total delay added by the short jobs to the completion times of the long jobs, and P_3 is the sum of the completion times of the long jobs calculated as if their schedule started at time 0. As $OPT = P_1 + P_2 + P_3$, we get $OPT = \frac{\gamma^2+1}{2}n^2 + \frac{1+\gamma}{2}n$. For the algorithm, we claim that it is best to schedule the processing part of each job right after its test. For short jobs, this is obvious, because a short job that was scheduled at a later time could be moved forward to right after its test without affecting the completion times of other jobs. This implies in particular that no two jobs complete at the same time, and that the completion times of any two jobs are at least one time unit apart. Furthermore, as it is clear that introducing idle time into the schedule cannot help, we can assume that all tests start and end at integral times and that all job completion times are integers. Now assume for a contradiction that some long job j is the first job for which the test completes at some time τ but the processing part completes at some time $\tau + k$ for $k > 1$. This implies that no job completes at time τ, and $r \leq k-1$ jobs have completion times in the interval $[\tau, \tau + k - 1]$. Moving job j forward to right after its test (and shifting all tests and job executions from time τ to $\tau + k - 1$ one time unit later) produces a schedule in which the completion time of job j decreases by $k-1$ while the completion times of only $r \leq k-1$ jobs increase by 1. Therefore, the modified schedule has an objective value that is the same or better. By repeating this transformation, we obtain a schedule where the processing part of each job is executed right after its test, without increasing the objective value. Therefore, executing the processing part of each job right after its test is the best the algorithm can do.

The objective value of the schedule produced by the agorithm is then $ALG = P_3 + P_2' + P_1$, where $P_2' = 2\gamma n(1-\gamma)n = 2\gamma(1-\gamma)n^2$ is the total delay that the long jobs with total length $2\gamma n$ add to the completion times of the $(1-\gamma)n$ short jobs. Thus, the objective value ALG of the schedule produded by the algorithm is $ALG = \frac{1+2\gamma-\gamma^2}{2}n^2 + \frac{1+\gamma}{2}n$. The competitive ratio, as a function of γ, is then:

$$\rho(\gamma) = \frac{ALG}{OPT} = \frac{\frac{1+2\gamma-\gamma^2}{2}n^2 + \frac{1+\gamma}{2}n}{\frac{\gamma^2+1}{2}n^2 + \frac{1+\gamma}{2}n} = \frac{1 + 2\gamma - \gamma^2 + \frac{1+\gamma}{n}}{\gamma^2 + 1 + \frac{1+\gamma}{n}}$$

For fixed γ, $\rho(\gamma)$ increases with n, and the ratio converges to $\frac{1+2\gamma-\gamma^2}{\gamma^2+1}$ for $n \to \infty$. The function $f(\gamma) = \frac{1+2\gamma-\gamma^2}{\gamma^2+1}$ has a global maximum at $\gamma_0 = \sqrt{2} - 1$ with $f(\gamma_0) = \sqrt{2}$, as can be shown using standard methods from calculus. Thus, if the adversary presents instances with arbitrarily large n and sets γ to the multiple of $\frac{1}{n}$ closest to $\sqrt{2} - 1$, it can force the algorithm to have competitive ratio arbitrarily close to $\sqrt{2}$. ◀

For solving the problem with uniform test times, we propose the algorithm SIDLE (Short Immediate, Delay Long Executions) that has a parameter $y > 0$. It tests all jobs, and executes a job j immediately after its test if $p_j \leq y$ (*short* job). The jobs j with $p_j > y$ (*long* jobs) are executed in SPT order at the end of the schedule, after all jobs have been tested and all short jobs executed. The algorithm is inspired by algorithm THRESHOLD from [5]. By adapting the analysis techniques from [5], we can show:

▶ **Theorem 16** (⋆). *Algorithm SIDLE with $y = y_0 \approx 1.35542$ has competitive ratio at most $\frac{1}{2}(1 - y_0 + y_0^2 + \sqrt{9 - 2y_0 - y_0^2 - 2y_0^3 + y_0^4}) \approx 1.58451 \leq 1.585$. Here, y_0 is the second root of the polynomial $2y^3 - 9y^2 + 10y - 2$.*

We remark that the analysis of Theorem 16 is tight: For $\alpha \approx 0.644584$ and $\gamma \approx 0.737781$, we can consider instances with $\alpha\gamma n$ jobs with processing time 0, $\alpha(1-\gamma)n$ jobs with processing time y_0, and $(1-\alpha)n$ jobs with processing time $y_0 + \epsilon$ for infinitesimally small ϵ. For large enough n, the competitive ratio of algorithm SIDLE on these instances is then approximately 1.58451.

5 Conclusion

In this paper, we have introduced a variant of scheduling with testing where every job must be tested and the objective is minimizing the sum of completion times. Our main result is an analysis showing that the competitive analysis of the 1-SORT algorithm is at most 1.861. For the special case of uniform test times, we have presented a 1.585-competitive algorithm as well as a lower bound of $\sqrt{2}$ on the competitive ratio of any deterministic algorithm.

There are several interesting directions for future research. First, there are gaps between our lower bound of $\sqrt{2}$ and our upper bounds of 1.585 and 1.861 on the competitive ratio for uniform and arbitrary test times, respectively. One immediate question is whether our analysis of 1-SORT can be improved, as we only know that the competitive ratio of 1-SORT is not better than 1.618. Our lower bound of $\sqrt{2}$ on the competitive ratio of deterministic algorithms holds for uniform test times; it would be interesting to find out whether the case of arbitrary test times admits a stronger lower bound. Furthermore, it would be worthwhile to study randomized algorithms for the problem. Finally, it would be interesting to explore whether our new technique for analyzing β-SORT can also be applied to other variants of scheduling with testing.

References

1 Susanne Albers and Alexander Eckl. Explorable uncertainty in scheduling with non-uniform testing times. In Christos Kaklamanis and Asaf Levin, editors, *Approximation and Online Algorithms - 18th International Workshop, WAOA 2020, Virtual Event, September 9-10, 2020, Revised Selected Papers*, volume 12806 of *Lecture Notes in Computer Science*, pages 127–142. Springer, 2020. doi:10.1007/978-3-030-80879-2_9.

2 Susanne Albers and Alexander Eckl. Scheduling with testing on multiple identical parallel machines. In Anna Lubiw and Mohammad R. Salavatipour, editors, *Algorithms and Data Structures - 17th International Symposium, WADS 2021, Virtual Event, August 9-11, 2021, Proceedings*, volume 12808 of *Lecture Notes in Computer Science*, pages 29–42. Springer, 2021. doi:10.1007/978-3-030-83508-8_3.

3 Evripidis Bampis, Konstantinos Dogeas, Alexander V. Kononov, Giorgio Lucarelli, and Fanny Pascual. Speed scaling with explorable uncertainty. In Kunal Agrawal and Yossi Azar, editors, *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*, pages 83–93. ACM, 2021. doi:10.1145/3409964.3461812.

4 Konstantinos Dogeas, Thomas Erlebach, and Ya-Chun Liang. Scheduling with obligatory tests. *CoRR*, abs/2406.16734, 2024. arXiv:2406.16734.

5 Christoph Dürr, Thomas Erlebach, Nicole Megow, and Julie Meißner. An adversarial model for scheduling with testing. *Algorithmica*, 82(12):3630–3675, 2020. doi:10.1007/s00453-020-00742-2.

6 Mingyang Gong, Zhi-Zhong Chen, and Kuniteru Hayashi. Approximation algorithms for multiprocessor scheduling with testing to minimize the total job completion time. *Algorithmica*, 2023. doi:10.1007/s00453-023-01198-w.

7 Mingyang Gong, Randy Goebel, Guohui Lin, and Eiji Miyano. Improved approximation algorithms for non-preemptive multiprocessor scheduling with testing. *J. Comb. Optim.*, 44(1):877–893, 2022. doi:10.1007/S10878-022-00865-Y.

8 Mingyang Gong and Guohui Lin. Improved approximation algorithms for multiprocessor scheduling with testing. In Jing Chen, Minming Li, and Guochuan Zhang, editors, *Frontiers of Algorithmics - International Joint Conference, IJTCS-FAW 2021, Beijing, China, August 16-19, 2021, Proceedings*, volume 12874 of *Lecture Notes in Computer Science*, pages 65–77. Springer, 2021. doi:10.1007/978-3-030-97099-4_5.

9 Retsef Levi, Thomas L. Magnanti, and Yaron Shaposhnik. Scheduling with testing. *Management Science*, 65(2):776–793, 2019. doi:10.1287/MNSC.2017.2973.

10 Alison Hsiang-Hsuan Liu, Fu-Hong Liu, Prudence W. H. Wong, and Xiao-Ou Zhang. The power of amortization on scheduling with explorable uncertainty. In Jaroslaw Byrka and Andreas Wiese, editors, *Approximation and Online Algorithms - 21st International Workshop, WAOA 2023, Amsterdam, The Netherlands, September 7-8, 2023, Proceedings*, volume 14297 of *Lecture Notes in Computer Science*, pages 90–103. Springer, 2023. doi:10.1007/978-3-031-49815-2_7.

11 Rajeev Motwani, Steven J. Phillips, and Eric Torng. Non-clairvoyant scheduling. *Theor. Comput. Sci.*, 130(1):17–47, 1994. doi:10.1016/0304-3975(94)90151-1.

12 Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956. doi:10.1002/nav.3800030106.