Explorable Uncertainty in Scheduling with Non-uniform Testing Times

Susanne Albers[1] and Alexander Eckl[1,2(✉)]

[1] Department of Informatics, Technical University of Munich, Boltzmannstr. 3, 85748 Garching, Germany
albers@in.tum.de, alexander.eckl@tum.de

[2] Advanced Optimization in a Networked Economy, Technical University of Munich, Arcisstraße 21, 80333 Munich, Germany

Abstract. The problem of scheduling with testing in the framework of explorable uncertainty models environments where some preliminary action can influence the duration of a task. In the model, each job has an unknown processing time that can be revealed by running a test. Alternatively, jobs may be run untested for the duration of a given upper limit. Recently, Dürr et al. [4] have studied the setting where all testing times are of unit size and have given lower and upper bounds for the objectives of minimizing the sum of completion times and the makespan on a single machine. In this paper, we extend the problem to non-uniform testing times and present the first competitive algorithms. The general setting is motivated for example by online user surveys for market prediction or querying centralized databases in distributed computing. Introducing general testing times gives the problem a new flavor and requires updated methods with new techniques in the analysis. We present constant competitive ratios for the objective of minimizing the sum of completion times in the deterministic case, both in the non-preemptive and preemptive setting. For the preemptive setting, we additionally give a first lower bound. We also present a randomized algorithm with improved competitive ratio. Furthermore, we give tight competitive ratios for the objective of minimizing the makespan, both in the deterministic and the randomized setting.

Keywords: Online scheduling · Explorable uncertainty · Competitive analysis · Single machine · Sum of completion times · Makespan

1 Introduction

In scheduling environments, uncertainty is a common consideration for optimization problems. Commonly, results are either based on worst case considerations or a random distribution over the input. These approaches are known as robust

Work supported by Deutsche Forschungsgemeinschaft (DFG), GRK 2201 and by the European Research Council, Grant Agreement No. 691672, project APEG.

C. Kaklamanis and A. Levin (Eds.): WAOA 2020, LNCS 12806, pp. 127–142, 2021.
https://doi.org/10.1007/978-3-030-80879-2_9

optimization and stochastic optimization, respectively. However, it is often the case that unknown information can be attained through investing some additional resources, e.g. time, computing power or money. In his seminal paper, Kahan [11] has first introduced the notion of explorable or queryable uncertainty to model obtaining additional information for a problem at a given cost during the runtime of an algorithm. Since then, these kind of problems have been explored in different optimization contexts, for example in the framework of combinatorial, geometric or function value optimization tasks.

Recently, Dürr et al. [4] have introduced a model for scheduling with testing on a single machine within the framework of explorable uncertainty. In their approach, a number of jobs with unknown processing times are given. Testing takes one unit of time and reveals the processing time. If a job is executed untested, the time it takes to run the job is given by an upper bound. The novelty of their approach lies in having tests executed directly on the machine running the jobs as opposed to considering tests separately.

In view of this model, a natural extension is to consider non-uniform testing times to allow for a wider range of problems. Dürr et al. state that for certain applications it is appropriate to consider a broader variation on testing times and leave this question up for future research.

Situations where a preliminary action, operation or test can be executed before a job are manifold and include a wide range of real-life applications. In the following, we discuss a small selection of such problems and emphasize cases with heterogeneous testing requirements. Consider first a situation where an online user survey can help predict market demand and production times. The time needed to produce the necessary amount of goods for the given demand is only known after conducting the survey. Depending on its scope and size, the invested costs for the survey may vary significantly.

As a second example, we look at distributed computing in a setting with many distributed local databases and one centralized master server. At the local stations, only estimates of some data values are stored; in order to obtain the true value one must query the master server. It depends on the distance and connection quality from any localized database to the master how much time and resources this requires. Olston and Widom [14] have considered this setting in detail.

Another possible example is the acquisition of a house through an agent giving us more information about its value, location, condition, etc., but demanding a price for her services. This payment could vary based on the price of the house, the amount of work of the agent or the number of competitors.

In their paper, Dürr et al. [4] mention fault diagnosis in maintenance and medical treatment, file compression for transmissions, and running jobs in an alternative fast mode whose availability can be determined through a test. Generally, any situation involving diverse cost and duration estimates, like e.g. in construction work, manufacturing or insurance, falls into our category of possible applications.

In view of all these examples, we investigate non-uniform testing in the scope of explorable uncertainty on a single machine as introduced by [4]. We study whether algorithms can be extended to this non-uniform case and if not, how we can find new methods for it.

1.1 Problem Statement

We consider n jobs to be scheduled on a single machine. Every job j has an unknown processing time p_j and a known upper bound u_j. It holds $0 \leq p_j \leq u_j$ for all j. Each job also has a testing time $t_j \geq 0$. A job can either be executed untested, which takes time u_j, or be tested and then executed, which takes a total time of $t_j + p_j$. Note that a tested job does not necessarily have to be executed right after its test, it may be delayed arbitrarily while the algorithm tests or executes other jobs.

Since only the upper bounds are initially known to the algorithm, the task can be viewed as an online problem with an adaptive adversary. The actual processing times p_j are only realized after job j has been tested by the algorithm. In the randomized case, the adversary knows the distribution of the random input parameters of an algorithm, but not their outcome.

We denote the completion time of a job j as C_j and primarily consider the objective of minimizing the total sum of completion times $\sum_j C_j$. As a secondary objective, we also investigate the simpler goal of minimizing the makespan $\max_j C_j$. We use competitive analysis to compare the value produced by an algorithm with an optimal offline solution.

Clearly, in the offline setting where all processing times are known, an optimal schedule can be determined directly: If $t_j + p_j \leq u_j$ then job j is tested, otherwise it is run untested. For the sum of completion times, the jobs are therefore scheduled in order of non-decreasing $\min(t_j + p_j, u_j)$. Any algorithm for the online problem not only has to decide whether to test a given job or not, but also in which order to run all tests and executions of both untested and tested jobs. For a solution to the makespan objective, the ordering of the jobs does not matter and an optimal offline algorithm decides the testing by the same principle as above.

1.2 Related Work

Our setting is directly based on the problem of scheduling uncertain jobs on a single machine with explorable processing times, introduced by Dürr et al. [4] in 2018. They only consider the special case where $t_j \equiv 1$ for all jobs. For deterministic algorithms, they give a lower bound of 1.8546 and an upper bound of 2. In the randomized case, they give a lower bound of 1.6257 and a 1.7453-competitive algorithm. For several deterministic special case instances, they provide upper bounds closer to the best possible ratio of 1.8546. Additionally, tight algorithms for the objective of minimizing the makespan are given for both the deterministic and randomized cases.

Testing and executing jobs on a single machine can be viewed as part of the research area of *queryable uncertainty* or *explorable uncertainty*. The first seminal paper on dealing with uncertainty by querying parts of the input was published in 1991 by Kahan [11]. In his paper, Kahan considers a set of elements with uncertain values that lie in a closed interval. He explores approximation guarantees for the number of queries necessary to obtain the maximum and median value of the uncertain elements.

Since then, there has been a large amount of research concerned with the objective of minimizing the number of queries to obtain a solution. A variety of numerical, geometric and combinatorial problems have been studied in this framework, the following is a selection of some of these publications: Next to Kahan, Feder et al. [8], Khanna and Tan [12], and Gupta et al. [10] have also considered the objective of determining different function values, in particular the k-smallest value and the median. Bruce et al. [2] have analysed geometric tasks, specifically the Maximal Points and Convex Hull problems. They have also introduced the notion of *witness sets* as a general concept for queryable uncertainty, which was then generalized by Erlebach et al. [6]. Olston and Widom [14] researched caching problems while allowing for some inaccuracy in the objective function. Other studied combinatorial problems include minimum spanning tree [6,13], shortest path [7], knapsack [9] and boolean trees [3]. See also the survey by Erlebach and Hoffmann [5] for an overview over research in this area.

A related type of problems within optimization under uncertainty are settings where the cost of the queries is a direct part of the objective function. Most notably, the paper by Dürr et al. [4] falls into this category. There, the tests necessary to obtain additional information about the runtime of the jobs are executed on the same machine as the jobs themselves. Other examples include Weitzman's original Pandora's Box problem [17], where n independent random variables are probed to maximize the highest revealed value. Every probing incurs a price directly subtracted from the objective function. Recently, Singla [16] introduced the 'price of information' model to describe receiving information in exchange for a probing price. He gives approximation ratios for various well-known combinatorial problems with stochastic uncertainty.

1.3 Contribution

In this paper, we provide the first algorithms for the more general scheduling with testing problem where testing times can be non-uniform. Consult Table 1 for an overview of results for both the non-uniform and uniform versions of the problem. All ratios provided without citation are introduced in this paper. The remaining results are presented in [4].

For the problem of scheduling uncertain jobs with non-uniform testing times on a single machine, our results are the following: A deterministic 4-competitive algorithm for the objective of minimizing the sum of completion times and a randomized 3.3794-competitive algorithm for the same objective. If we allow preemption - that is, to cancel the execution of a job at any time and start

Table 1. Overview of results

Objective type	General tests	Uniform tests	Lower bound
$\sum C_j$ - deterministic	4	2 [4]	1.8546 [4]
$\sum C_j$ - randomized	3.3794	1.7453 [4]	1.6257 [4]
$\sum C_j$ - determ. preemptive	$2\varphi \approx 3.2361$	-	1.8546
$\max C_j$ - deterministic	$\varphi \approx 1.6180$	φ [4]	φ [4]
$\max C_j$ - randomized	$\frac{4}{3}$	$\frac{4}{3}$ [4]	$\frac{4}{3}$ [4]

working on a different job - then we can improve the deterministic case to be 2φ-competitive. Here, $\varphi \approx 1.6180$ is the golden ratio.

For the objective of minimizing the makespan, we adopt and extend the ideas of Dürr et al. [4] to provide a tight φ-competitive algorithm in the deterministic case and a tight $\frac{4}{3}$-competitive algorithm in the randomized case.

Our approaches handle non-uniform testing times in a novel fashion distinct from the methods of [4]. As we show in the full version of this paper [1], the idea of scheduling untested jobs with small upper bounds in the beginning of the schedule, which works well in the uniform case, fails to generalize to non-uniform tests. Additionally, describing parameterized worst-case instances becomes intangible in the presence of an arbitrary number of different testing times.

In place of these methods, we compute job completion times by cross-examining contributions of other jobs in the schedule. We determine tests based on the ratio between the upper bound and the given test time and pay specific attention to sorting the involved executions and tests in an suitable way.

The paper is structured as follows: Sects. 2 and 3 examine the deterministic and randomized cases respectively. Various algorithms are presented and their competitive ratios proven. We extend the optimal results for the objective of minimizing the makespan from the uniform case to general testing times in Sect. 4. Finally, we conclude with some open problems.

2 Deterministic Setting

In this section, we introduce our basic algorithm and prove deterministic upper bounds for the non-preemptive as well as the preemptive case. The basic structure introduced in Sect. 2.1 works as a framework for other algorithms presented later. We give a detailed analysis of the deterministic algorithm and prove that it is 4-competitive if parameters are chosen accordingly. In Sect. 2.2 we prove that an algorithm for the preemptive case is 3.2361-competitive and that no preemptive algorithm can have a ratio better than 1.8546.

2.1 Basic Algorithm and Proof of 4-Competitiveness

We now present the elemental framework of our algorithm, which we call (α, β)-*SORT*. As input, the algorithm has two real parameters, $\alpha \geq 1$ and $\beta \geq 1$.

Algorithm 1: (α, β)-SORT

```
T ← ∅, N ← ∅, σ_j ≡ 0;
foreach j ∈ [m] do
    if u_j ≥ α t_j then
        add j to T;
        set σ_j ← β t_j;
    else
        add j to N;
        set σ_j ← u_j;
    end
end
while N ∪ T ≠ ∅ do
    choose j_min ∈ argmin_{j∈N∪T} σ_j;
    if j_min ∈ N then
        execute j_min untested;
        remove j_min from N;
    else if j_min ∈ T then
        if j_min not tested then
            test j_min;
            set σ_{j_min} ← p_{j_min};
        else
            execute j_min;
            remove j_min from T;
        end
end
```

The algorithm is divided into two phases. First, we decide for each job whether we test this job or not based on the ratio $\frac{u_j}{t_j}$. This gives us a partition of $[m]$ into the disjoint sets $T = \{j \in [m] : \text{ALG tests } j\}$ and $N = \{j \in [m] : \text{ALG runs } j \text{ untested}\}$. In the second phase, we always attend to the job $j_{\min}$ with the current smallest *scaling time* σ_j. The scaling time is the time needed for the next step of executing j:

- If j is in N, then $\sigma_j = u_j$.
- If j is in T and has not been tested, then $\sigma_j = \beta t_j$.
- If j is in T and has already been tested, then $\sigma_j = p_j$.

Note that in the second case above, we 'stretch' the scaling time by multiplying with $\beta \geq 1$. The intention behind this stretching is that testing a job, unlike executing it, does not immediately lead to a job being completed. Therefore the parameter β artificially lowers the relevance of testing in the ordering of our algorithm. Note that the actual time needed for testing remains t_j.

In the following, we show that the above algorithm achieves a provably good competitive ratio. The parameters are kept general in the proof and are then optimized in a final step. We present the computations with general parameters for a clearer picture of the proof structure, which we will reuse in later sections.

In the final optimization step it will turn out that setting $\alpha = \beta = 1$ yields a best-possible competitive ratio of 4.

Theorem 1. *The $(1,1)$-SORT algorithm is 4-competitive for the objective of minimizing the sum of completion times.*

Proof. For the purpose of estimating the algorithmic result against the optimum, let $\rho_j := \min(u_j, t_j + p_j)$ be the optimal running time of job j. Without loss of generality, we order the jobs s.t. $\rho_1 \geq \ldots \geq \rho_n$. Hence the objective value of the optimum is

$$\mathrm{OPT} = \sum_{j=1}^{n} j \cdot \rho_j \tag{1}$$

Additionally, let

$$p_j^A := \begin{cases} t_j + p_j & \text{if } j \in T, \\ u_j & \text{if } j \in N, \end{cases} \tag{2}$$

be the *algorithmic running time* of j, i.e. the time the algorithm spends on running job j.

We start our analysis by comparing p_j^A to the optimal runtime ρ_j for a single job, summarized in the following Proposition:

Proposition 1. *(a)* $\forall j \in T$: $t_j \leq \rho_j$, $p_j \leq \rho_j$
(b) $\forall j \in T$: $p_j^A \leq \left(1 + \frac{1}{\alpha}\right) \rho_j$
(c) $\forall j \in N$: $p_j^A \leq \alpha \rho_j$

Part (a) directly estimates testing and running times of tested jobs against the values of the optimum. We will use this extensively when computing the completion time of the jobs. The proof of parts (b) and (c) is very similar to the proof of Theorem 14 in [4] for uniform testing times. We refer to the full version [1] for a complete write-down of the proof. Note that instead of considering a single bound, we split the upper bound of the algorithmic running time p_j^A into different results for tested (b) and untested jobs (c). This allows us to differentiate between different cases in the proof of Lemma 1 in more detail. We will often make use of this Proposition to upper bound the algorithmic running time in later sections.

To obtain an estimate of the completion time C_j, we consider the *contribution* $c(k,j)$ of all jobs $k \in [n]$ to C_j. We define $c(k,j)$ to be the amount of time the algorithm spends scheduling job k before the completion of j. Obviously it holds that $c(k,j) \leq p_k^A$. The following central lemma computes an improved upper bound on the contribution $c(k,j)$, using a rigorous case distinction over all possible configurations of k and j:

Lemma 1 (Contribution Lemma). *Let $j \in [n]$ be a given job. The completion time of j can be written as*

$$C_j = \sum_{k \in [n]} c(k,j).$$

Additionally, for the contribution of k to j it holds that

$$c(k,j) \le \max\left(\left(1+\frac{1}{\beta}\right)\alpha, 1+\frac{1}{\alpha}, 1+\beta\right)\rho_j.$$

Refer to the full version [1] for the proof. Depending on whether j and k are tested or not, the lemma computes various upper bounds on the contribution using estimates from Proposition 1. Finally, the given bound on $c(k,j)$ is achieved by taking the maximum over the different cases.

Recall that the jobs are ordered by non-increasing optimal execution times ρ_j, which by Proposition 1 are directly tied to the algorithmic running times. Hence, the jobs k with small indices are the 'bad' jobs with possibly large running times. For jobs with $k \le j$ we therefore use the independent upper bound from the Contribution Lemma. Jobs with large indices $k > j$ are handled separately and we directly estimate them using their running time p_k^A.

By Lemma 1 and Proposition 1(b),(c) we have

$$\begin{aligned}
C_j &= \sum_{k>j} c(k,j) + \sum_{k\le j} c(k,j) \\
&\le \sum_{k>j} p_k^A + \sum_{k\le j} \max\left(\left(1+\frac{1}{\beta}\right)\alpha, 1+\frac{1}{\alpha}, 1+\beta\right)\rho_j \\
&= \sum_{k>j} \max\left(\alpha, 1+\frac{1}{\alpha}\right)\rho_k + \max\left(\left(1+\frac{1}{\beta}\right)\alpha, 1+\frac{1}{\alpha}, 1+\beta\right) j\cdot\rho_j.
\end{aligned}$$

Finally, we sum over all jobs j:

$$\begin{aligned}
\sum_{j=1}^{n} C_j &= \sum_{j=1}^{n}\sum_{k=j+1}^{n} \max\left(\alpha, 1+\frac{1}{\alpha}\right)\rho_k \\
&\qquad + \sum_{j=1}^{n} \max\left(\left(1+\frac{1}{\beta}\right)\alpha, 1+\frac{1}{\alpha}, 1+\beta\right) j\cdot\rho_j \\
&= \max\left(\alpha, 1+\frac{1}{\alpha}\right)\sum_{j=1}^{n}(j-1)\rho_j \\
&\qquad + \max\left(\left(1+\frac{1}{\beta}\right)\alpha, 1+\frac{1}{\alpha}, 1+\beta\right)\sum_{j=1}^{n} j\cdot\rho_j \\
&\le \underbrace{\left(\max\left(\alpha, 1+\frac{1}{\alpha}\right) + \max\left(\left(1+\frac{1}{\beta}\right)\alpha, 1+\frac{1}{\alpha}, 1+\beta\right)\right)}_{=:f(\alpha,\beta)}\sum_{j=1}^{n} j\cdot\rho_j \\
&= f(\alpha,\beta)\cdot \mathrm{OPT}
\end{aligned}$$

Minimizing $f(\alpha,\beta)$ on the domain $\alpha,\beta \ge 1$ yields optimal parameters $\alpha=\beta=1$ and a value of $f(1,1)=4$. We conclude that $(1,1)$-SORT is 4-competitive.

The parameter selection $\alpha = 1$, $\beta = 1$ is optimal for the closed upper bound formula we obtained in our proof. It is possible and somewhat likely that a different parameter choice leads to better overall results for the algorithm. In the optimal makespan algorithm (see Sect. 4) the value of α is higher, suggesting that $\alpha = 1$, which leads to testing all non-trivial jobs, might not be the best choice. The problem structure and the approach by Dürr et al. [4] also motivate setting β to some higher value than 1. For our proof, setting parameters like we did is optimal.

In the full version of the paper [1], we take advantage of this somewhat unexpected parameter outcome to prove that $(1, 1)$-SORT cannot be better than 3-competitive. Additionally, we show that for *any* choice of parameters, (α, β)-SORT is not better than 2-competitive.

2.2 A Deterministic Algorithm with Preemption

The goal of this section is to show that if we allow jobs to be preempted there exists a 3.2361-competitive algorithm. In his book on Scheduling, Pinedo [15] defines preemption as follows: "The scheduler is allowed to interrupt the processing of a job (preempt) at any point in time and put a different job on the machine instead."

The idea for our algorithm in the preemptive setting is based on the so-called *Round Robin* rule, which is used frequently in preemptive machine scheduling [15, Chapters 3.7, 5.6, 12.4]. The scheduling time frame is divided into very small equal-sized units. The Round Robin algorithm then cycles through all jobs, tending to each job for exactly one unit of time before switching to the next. It ensures that at any time the amount every job has been processed only differs by at most one time unit [15].

The Round Robin algorithm is typically applied when job processing times are completely unknown. In our setting, we are actually given some upper bounds for our processing times and may invest testing time to find out the actual values. Despite having more information, it turns out that treating all job processing times as unknown in a Round Robin setting gives a provably good result. The only way we employ upper bounds and testing times is again to decide which jobs will be tested and which will not. We again do this at the beginning of our schedule for all given jobs. The rule to decide testing is exactly the same as in the first phase of Algorithm 1: If $u_j/t_j \geq \alpha$, then test j, otherwise run j untested. Again, α is a parameter that is to be determined. It will turn out that setting $\alpha = \varphi$ gives the best result.

The pseudo-code for the *Golden Round Robin* algorithm is given in Algorithm 2.

Essentially, the algorithm first decides for all jobs whether to test them and then runs a regular Round Robin scheme on the algorithmic testing time p_j^A, which is defined as in (2).

Theorem 2. *The Golden Round Robin algorithm is* 3.2361*-competitive in the preemptive setting for the objective of minimizing the sum of completion times. This analysis is tight.*

Algorithm 2: Golden Round Robin

```
T ← ∅, N ← ∅, σ_j ≡ 0;
foreach j ∈ [m] do
    if u_j ≥ φ t_j then
        add j to T;
        set σ_j ← t_j;
    else
        add j to N;
        set σ_j ← u_j;
    end
end
while ∃j ∈ [m] not completely scheduled do
    run Round Robin on all jobs using σ_j as their processing time;
    let j_min be the first job to finish during the current execution;
    if j_min ∈ T and j_min tested but not executed then
        set σ_{j_min} ← p_{j_min} and keep j_min in the Round Robin rotation;
    end
end
```

We only provide a sketch of the proof here, the complete proof can be found in the full version of the paper [1].

Proof (Proof sketch). We set $\alpha = \varphi$ and use Proposition 1(b),(c) to bound the algorithmic running time p_j^A of a job j by its optimal running time ρ_j.

$$p_j^A \leq \varphi \rho_j.$$

We then compute the contribution of a job k to a fixed job j by grouping jobs based on their finishing order in the schedule. This allows us to estimate the completion time of job j:

$$C_j \leq \sum_{k>j} p_k^A + j \cdot p_j^A$$

Finally, we sum over all jobs to receive $\text{ALG} \leq 2\varphi \cdot \text{OPT}$.

To show that the analysis is tight, we provide an example where the algorithmic solution has a value of $2\varphi \cdot \text{OPT}$ if we let the number of jobs approach infinity.

The following theorem additionally provides a lower bound for the deterministic preemptive setting, giving us a first simple lower bound for this case. The proof is based on the lower bound provided in [4] for the deterministic non-preemptive case. We again defer the proof to the full version [1].

Theorem 3. *No algorithm in the preemptive deterministic setting can be better than* 1.8546*-competitive.*

3 Randomized Setting

In this section we introduce randomness to further improve the competitive ratio of Algorithm 1. There are two natural places to randomize: when deciding which jobs to test and the decision about the ordering of the jobs. These decisions directly correspond to the parameters α and β.

Making α randomized, for instance, could be achieved by defining α as a random variable with density function $f_\alpha : [1, \infty] \to \mathbb{R}_0^+$ and testing j if and only if $r_j := u_j/t_j \geq \alpha$. Then the probability for testing j would be given by $p = \int_1^{r_j} f_\alpha(x)dx$. Using a random variable α like this would make the analysis unnecessarily complicated, therefore we directly consider the probability p without defining a density, and let p depend on r_j. This additionally allows us to compute the probability of testing *independently* for each job.

Introducing randomness for β is even harder. The choice of β influences multiple jobs at the same time, therefore independence is hard to establish. Additionally, β appears in the denominator of our analysis frequently, hindering computations using expected values. We therefore forgo using randomness for the β-parameter and focus on α in this paper. We encourage future research to try their hand at making β random.

We give a short pseudo-code of our randomized algorithm in Algorithm 3. It is given a parameter-function $p(r_j)$ and a parameter β, both of which are to be determined later.

Algorithm 3: Randomized-SORT

```
T ← ∅, N ← ∅, σ_j ≡ 0;
foreach j ∈ [m] do
    add j to T with probability p(r_j) and set σ_j ← βt_j;
    otherwise add it to N and set σ_j ← u_j;
end
while N ∪ T ≠ ∅ do
    choose j_min ∈ argmin_{j∈N∪T} σ_j;
    if j_min ∈ N then
        execute j_min untested;
        remove j_min from N;
    else if j_min ∈ T then
        if j_min not tested then
            test j_min;
            set σ_{j_min} ← p_{j_min};
        else
            execute j_min;
            remove j_min from T;
        end
end
```

Theorem 4. *Randomized-SORT is* 3.3794*-competitive for the objective of minimizing the sum of completion times.*

Proof. Again, we let $\rho_1 \geq \ldots \geq \rho_n$ denote the ordered optimal running time of jobs $1, \ldots, n$. The optimal objective value is given by (1). Fix jobs j and k. For easier readability, we write p instead of $p(r_j)$. Since the testing decision is now done randomly, the algorithmic running time p_j^A as well as the contribution $c(k, j)$ are now random variables. It holds

$$p_j^A = \begin{cases} t_j + p_j & \text{with probability } p \\ u_j & \text{with probability } 1 - p \end{cases}$$

For the values of $c(k, j)$ we consult the case distinctions from the proof of the Contribution Lemma 1. If $j \in N$, one can easily determine that $c(k, j) \leq (1 + 1/\beta)u_j$ for all cases. Note that for this we did not need to use the final estimates with parameter α from the case distinction. Therefore this upper bound holds deterministically as long as we assume $j \in N$. By extension it also trivially holds for the expectation of $c(k, j)$:

$$E[c(k, j) \,|\, j \text{ untested}] \leq (1 + 1/\beta)u_j.$$

Doing the same for the case distinction of $j \in T$, we get

$$E[c(k, j) \,|\, j \text{ tested}] \leq \max\left((1 + \beta)t_j, \left(1 + \frac{1}{\beta}\right)p_j, t_j + p_j\right).$$

For the expected value of the contribution we have by the law of total expectation:

$$\begin{aligned} E[c(k, j)] &= E[c(k, j) \,|\, j \text{ untested}] \cdot Pr[j \text{ untested}] \\ &\quad + E[c(k, j) \,|\, j \text{ tested}] \cdot Pr[j \text{ tested}] \\ &\leq \left(1 + \frac{1}{\beta}\right)u_j \cdot (1 - p) + \max\left((1 + \beta)t_j, \left(1 + \frac{1}{\beta}\right)p_j, t_j + p_j\right) \cdot p \end{aligned}$$

Note that this estimation of the expected value is independent of any parameters of k. That means, for fixed j we estimate the contribution to be the same for all jobs with small parameter $k \leq j$. Of course, as before, for the jobs with large parameter $k > j$ we may also alternatively directly use the algorithmic runtime of k:

$$E[c(k, j)] \leq E[p_k^A].$$

Putting the above arguments together, we use the Contribution Lemma and linearity of expectation to estimate the completion time of j:

$$\begin{aligned} E[C_j] &= \sum_{j=1}^{n} E[c(k, j)] \\ &\leq \sum_{k>j} E[p_k^A] + \sum_{k \leq j} E[c(k, j)]. \end{aligned}$$

For the total objective value of the algorithm we receive again using linearity of expectation:

$$\begin{aligned} E\left[\sum_{j=1}^{n} C_j\right] &\leq \sum_{j=1}^{n}(j-1)E[p_j^A] + \sum_{j=1}^{n} j \cdot E[c(k,j)] \\ &\leq \sum_{j=1}^{n}(j-1)(u_j \cdot (1-p) + (t_j + p_j) \cdot p) \\ &\quad + \sum_{j=1}^{n} j\Bigg(\left(1+\frac{1}{\beta}\right) u_j \cdot (1-p) \\ &\quad + \max\left((1+\beta)t_j, \left(1+\frac{1}{\beta}\right) p_j, t_j + p_j\right) \cdot p\Bigg) \\ &\leq \sum_{j=1}^{n} j \cdot \lambda_j(\beta, p), \end{aligned}$$

where we define

$$\begin{aligned} \lambda_j(\beta,p) := &\left(u_j + \left(1+\frac{1}{\beta}\right) u_j\right) \cdot (1-p) \\ &+ \left(t_j + p_j + \max\left((1+\beta)t_j, \left(1+\frac{1}{\beta}\right) p_j, t_j + p_j\right)\right) \cdot p. \end{aligned}$$

Having computed this first estimation for the objective of the algorithm, we now consider the ratio $\lambda_j(\beta,p)/\rho_j$ as a standalone. If we can prove an upper bound for this ratio, the same holds as competitive ratio for our algorithm.

Hence the goal is to choose parameters β and p, where p can depend on j, s.t. $\lambda_j(\beta,p)/\rho_j$ is as small as possible. In the best case, we want to compute

$$\min_{\beta \geq 1, p \in [0,1]} \max_j \frac{\lambda_j(\beta,p)}{\rho_j}.$$

Lemma 2. *There exist parameters $\hat{\beta} \geq 1$ and $\hat{p} \in [0,1]$ s.t.*

$$\max_j \frac{\lambda_j(\hat{\beta},\hat{p})}{\rho_j} \leq 3.3794.$$

The choice of parameters is given in the proof of the lemma, which can be found in the full version of our paper [1]. During the proof we use computer-aided computations with Mathematica. The Mathematica code can be found in the full version.

To conclude the proof of the theorem, we write

$$E\left[\sum_{j=1}^{n} C_j\right] \leq \sum_{j=1}^{n} j \cdot \lambda_j(\hat{\beta},\hat{p}) \leq 3.3794 \sum_{j=1}^{n} j \cdot \rho_j = 3.3794 \cdot \mathrm{OPT}.$$

4 Optimal Results for Minimizing the Makespan

In this section, we consider the objective of minimizing the makespan of our schedule. It turns out that we are able to prove the same tight algorithmic bounds for this objective function as Dürr et al. in the unit-time testing case, both for deterministic and randomized algorithms. The decisions of the algorithms only depend on the ratio $r_j = u_j/t_j$. Refer to the full version [1] for the proofs.

Theorem 5. *The algorithm that tests job j iff $r_j \geq \varphi$ is φ-competitive for the objective of minimizing the makespan. No deterministic algorithm can achieve a smaller competitive ratio.*

Theorem 6. *The randomized algorithm that tests job j with probability $p = 1-1/(r_j^2-r_j+1)$ is $4/3$-competitive for the objective of minimizing the makespan. No randomized algorithm can achieve a smaller competitive ratio.*

5 Conclusion

In this paper, we introduced the first algorithms for the problem of scheduling with testing on a single machine with general testing times that arises in the context of settings where a preliminary action can influence cost, duration or difficulty of a task. For the objective of minimizing the sum of completion times, we presented a 4-approximation for the deterministic case, and a 3.3794-approximation for the randomized case. If preemption is allowed, we can improve the deterministic result to 3.2361. We also considered the objective of minimizing the makespan, for which we showed tight ratios of 1.618 and 4/3 for the deterministic and randomized cases, respectively.

Our results open promising avenues for future research, in particular tightening the gaps between our ratios and the lower bounds given by the unit case. Based on various experiments using different adversarial behaviour and multiple testing times it seems hard to force the algorithm to make mistakes that lead to worse ratios than those proven in [4] for the unit case. We conjecture that in order to achieve better lower bounds, the adversary must make live decisions based on previous choices of the algorithm, in particular depending on how much the algorithm has already tested, run or deferred jobs up to a certain point.

Further interesting directions for future work are the extension of the problem to multiple machines to consider scheduling problems like open shop, flow shop, or other parallel machine settings.

References

1. Albers, S., Eckl, A.: Explorable uncertainty in scheduling with non-uniform testing times (2020). https://arxiv.org/abs/2009.13316
2. Bruce, R., Hoffmann, M., Krizanc, D., Raman, R.: Efficient update strategies for geometric computing with uncertainty. Theor. Comput. Syst. **38**(4), 411–423 (2005). https://doi.org/10.1007/s00224-004-1180-4

3. Charikar, M., Fagin, R., Guruswami, V., Kleinberg, J., Raghavan, P., Sahai, A.: Query strategies for priced information. J. Comput. Syst. Sci. **64**(4), 785–819 (2002). https://doi.org/10.1006/jcss.2002.1828
4. Dürr, C., Erlebach, T., Megow, N., Meißner, J.: Scheduling with explorable uncertainty. In: Karlin, A.R. (ed.) 9th Innovations in Theoretical Computer Science Conference (ITCS 2018). Leibniz International Proceedings in Informatics (LIPIcs), vol. 94, pp. 30:1–30:14. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2018). https://doi.org/10.4230/LIPIcs.ITCS.2018.30
5. Erlebach, T., Hoffmann, M.: Query-competitive algorithms for computing with uncertainty. Bull. EATCS (116), 22–39 (2015)
6. Erlebach, T., Hoffmann, M., Krizanc, D., Mihal'ák, M., Raman, R.: Computing minimum spanning trees with uncertainty. In: Albers, S., Weil, P. (eds.) 25th International Symposium on Theoretical Aspects of Computer Science. Leibniz International Proceedings in Informatics (LIPIcs), vol. 1, pp. 277–288. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2008). https://doi.org/10.4230/LIPIcs.STACS.2008.1358
7. Feder, T., Motwani, R., O'Callaghan, L., Olston, C., Panigrahy, R.: Computing shortest paths with uncertainty. J. Algorithms **62**(1), 1–18 (2007). https://doi.org/10.1016/j.jalgor.2004.07.005
8. Feder, T., Motwani, R., Panigrahy, R., Olston, C., Widom, J.: Computing the median with uncertainty. SIAM J. Comput. **32**(2), 538–547 (2003). https://doi.org/10.1137/S0097539701395668
9. Goerigk, M., Gupta, M., Ide, J., Schöbel, A., Sen, S.: The robust knapsack problem with queries. Comput. Oper. Res. **55**, 12–22 (2015). https://doi.org/10.1016/j.cor.2014.09.010
10. Gupta, M., Sabharwal, Y., Sen, S.: The update complexity of selection and related problems. In: Chakraborty, S., Kumar, A. (eds.) IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011). Leibniz International Proceedings in Informatics (LIPIcs), vol. 13, pp. 325–338. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2011). https://doi.org/10.4230/LIPIcs.FSTTCS.2011.325
11. Kahan, S.: A model for data in motion. In: Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing, pp. 265–277. STOC 1991. Association for Computing Machinery, New York, NY, USA (1991). https://doi.org/10.1145/103418.103449
12. Khanna, S., Tan, W.C.: On computing functions with uncertainty. In: Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 171–182. PODS 2001. Association for Computing Machinery, New York, NY, USA (2001). https://doi.org/10.1145/375551.375577
13. Megow, N., Meißner, J., Skutella, M.: Randomization helps computing a minimum spanning tree under uncertainty. SIAM J. Comput. **46**(4), 1217–1240 (2017). https://doi.org/10.1137/16M1088375
14. Olston, C., Widom, J.: Offering a precision-performance tradeoff for aggregation queries over replicated data. In: 26th International Conference on Very Large Data Bases (VLDB 2000), pp. 144–155. VLDB 2000. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2000)
15. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems. Springer International Publishing, 5 edn. (2016). https://doi.org/10.1007/978-1-4614-2361-4

16. Singla, S.: The price of information in combinatorial optimization. In: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 2523–2532. SODA 2018, Society for Industrial and Applied Mathematics, USA (2018)
17. Weitzman, M.L.: Optimal search for the best alternative. Econometrica **47**(3), 641–654 (1979). https://doi.org/10.2307/1910412