Noname manuscript No.
(will be inserted by the editor)

An Adversarial Model for Scheduling with Testing

Christoph Dürr · Thomas Erlebach · Nicole Megow · Julie Meißner

the date of receipt and acceptance should be inserted later

Abstract We introduce a novel adversarial model for scheduling with explorable uncertainty. In this model, the processing time of a job can potentially be reduced (by an *a priori* unknown amount) by testing the job. Testing a job j takes one unit of time and may reduce its processing time from the given upper limit $\bar{p}_j$ (which is the time taken to execute the job if it is not tested) to any value between 0 and $\bar{p}_j$. This setting is motivated e.g. by applications where a code optimizer can be run on a job before executing it. We consider the objective of minimizing the sum of completion times on a single machine. All jobs are available from the start, but the reduction in their processing times as a result of testing is unknown, making this an online problem that is amenable to competitive analysis. The need to balance the time spent on tests and the time spent on job executions adds a novel flavor to the problem. We give the first and nearly tight lower and upper bounds on the competitive ratio for deterministic and randomized algorithms. We also show that minimizing the makespan is a consid-

This research was carried out in the framework of Matheon supported by Einstein Foundation Berlin, the German Science Foundation (DFG) under contract ME 3825/1 and Bayerisch-Französisches Hochschulzentrum (BFHZ). Further support was provided by EPSRC grant EP/S033483/1 and the ANR grant ANR-18-CE25-0008. The second author was supported by a study leave granted by University of Leicester during the early stages of the research. A preliminary version of this paper appeared in *The 9th Innovations in Theoretical Computer Science Conference* (ITCS), January 2018 [16].

Corresponding author

Christoph Dürr
Sorbonne Université, CNRS, Laboratoire d'informatique de Paris 6, Paris, France. E-mail: Christoph.Durr@LIP6.fr. ORCID: 0000-0001-8103-5333.

Thomas Erlebach
School of Informatics, University of Leicester, UK. E-mail: te17@leicester.ac.uk. ORCID: 0000-0002-4470-5868.

Nicole Megow
Department of Mathematics and Computer Science, University of Bremen, Germany. E-mail: nicole.megow@uni-bremen.de. ORCID: 0000-0002-3531-7644

Julie Meißner
Institute of Mathematics, Technical University of Berlin, Germany. E-mail: jmeiss@math.tu-berlin.de

erably easier problem for which we give optimal deterministic and randomized online algorithms.

Keywords Explorable Uncertainty · Competitive Analysis · Lower Bounds · Scheduling

1 Introduction

Uncertainty in scheduling has been modeled and investigated in many different ways, particularly in the frameworks of online optimization, stochastic optimization, and robust optimization. All these different approaches have the common assumption that the uncertain information, e.g., the processing time of a job, cannot be explored before making scheduling decisions. However, in many applications there is the opportunity to gain exact or more precise information at a certain additional cost, e.g., by investing time, money, or energy. It is a challenging problem to design algorithms that balance the cost for data exploration and the benefit for the quality of a solution. This involves quantifying the trade-off between exploration and exploitation, as it is ubiquitous in numerous applications.

In this paper, we introduce a novel model for scheduling with explorable uncertainty. Given a set of n jobs, every job j can optionally be *tested* prior to its execution. A job that is executed without testing has processing time $\bar{p}_j \geq 0$, while a tested job has processing time p_j with $0 \leq p_j \leq \bar{p}_j$. Testing a job takes one unit of time on the same resource (machine) that processes jobs. A tested job does not need to be executed right after its test.

Initially the algorithm knows for each job j only the upper limit $\bar{p}_j$, and gets to know the time p_j only after a test. Tested jobs can be executed at any time after their test. An algorithm must carefully balance testing and execution of jobs by evaluating the benefit and cost for testing. The resulting schedule is constructed by the algorithm adaptively. This means that at every moment, the choice of a job, and the decision whether to execute or to test it, may depend on the outcome of previous tests.

We focus on scheduling on a single machine. Unless otherwise noted, we consider the sum of completion times as the minimization objective. We use competitive analysis to assess the performance of algorithms.

For the standard version of this single-machine scheduling problem, i.e., without testing, it is well known that the Shortest Processing Time (SPT) rule is optimal for minimizing the sum of completion times. The addition of testing, combined with the fact that the processing times p_j are initially unknown to the algorithm, turns the problem into an online problem with a novel flavor. An algorithm must decide which jobs to execute untested and which jobs to test. Once a job has been tested, the algorithm must decide whether to execute it immediately or to defer its execution while testing or executing other jobs. At any point in the schedule, it may be difficult to choose between testing a job (which might reveal that it has a very short processing time and hence is ideally suited for immediate execution) and executing an untested or previously tested job. Testing a job yields information that may be useful for the scheduler, but may delay the completion times of many jobs. Finding the right balance between tests and executions poses an interesting challenge.

1.1 Motivation and applications

Scheduling with testing is motivated by a range of application settings where an information-revealing test can be applied to jobs before they are executed which leads to a trade-off regarding how to allocate resources for performing a test and actually executing the job. We discuss some examples of such settings from very different domains.

First, consider the execution of computer programs on a processor. A test could correspond to a code optimizer that takes unit time to process the program and potentially reduces its running-time. The upper limit of a job describes the running-time of the program if the code optimizer is not executed. See [10, Chapter 5] for an overview of various code otimization techniques.

Second, consider the transmission of files over a network link. It is possible to run a compression algorithm that can reduce the size of a file by an *a priori* unknown amount. If a file is incompressible (e.g., if it is already compressed), its size cannot be reduced at all. Running the compression algorithm corresponds to a test. See [54, 59] for some practical techniques balancing compression time with transmission time.

An algorithmic application concerns jobs, which can be executed in two different modes, a *safe* mode and an *alternative* mode. The safe mode is always possible. The alternative mode may have a shorter processing time, but is not possible for every job. A test is necessary to determine whether the alternative mode is possible for a job and what the processing time in the alternative mode would be. One example would be computing shortest paths in several given graphs. Solving it in the safe mode would involve the Bellman-Ford algorithm, while the faster alternative mode uses Dijkstra's algorithm requiring a preliminary non-negativity test on the edge weights. This situation would be faced by a server that solves shortest paths problems submitted by users. See [34] for a survey on algorithm selection techniques.

As a final application area consider scenarios, where a diagnosis can be carried out to determine the exact processing time of a job. This is the case in very different domains such as diagnostics in *maintenance* or in *medical environments* such as emergency departments. A fault diagnosis can determine the time needed to repair or replace a device, which allows for an efficient schedule of maintenance operations. There is a vast amount of literature on maintenance models; see e.g. [49, 51]. In medical diagnostics, information can be acquired about the time needed for consultation, treatment session and other activities with the patient. This information can help to prioritize and efficiently allocate limited medical resources; cf. [2, 39, 46]. Assuming that the resource that performs the diagnosis is the same resource that executes the job, e.g., an engineer or a medical doctor, we are in our problem setting of scheduling with testing with the trade-off regarding how to allocate resources between diagnostics and actual execution of jobs.

In some applications, it may be appropriate to allow the time for testing a job to be different for different jobs (e.g., proportional to the upper limit of a job). Furthermore, there are applications where the job processing time is p_j even if executed untested, and the test reveals p_j, which otherwise is only known to belong to the interval $[0, \bar{p}_j]$. We leave the consideration of such generalizations of the problem to future work.

competitive ratio	lower bounds	upper bounds
deterministic algorithms	1.8546 (Thm 9)	2 THRESHOLD (Thm 7)
randomized algorithms	1.6257 (Thm 11)	1.7453 RANDOM (Thm 10)
uniform instances (det)	1.8546 (Thm 9)	1.9338* BEAT (Thm 12)
extreme uniform instances (det)	1.8546 (Thm 9)	1.8668 UTE (Thm 18)
extreme uniform with $\bar{p} \approx 1.989$ (det)	1.8546 (Thm 9)	1.8552 UTE (Cor 19)

Table 1 Our results for minimizing the sum of completion times. * holds asymptotically

1.2 Our contribution

A scheduling algorithm in the model of explorable uncertainty has to make two types of decisions: which jobs should be tested, and in what order should job executions and tests be scheduled. There is a subtle compromise to be found between investing time to test jobs and the benefit one can gain from these tests. We design scheduling algorithms that address this exploration-exploitation question in different ways and provide nearly tight bounds on the competitive ratio. In our analysis, we first show that worst-case instances have a particular structure that can be described by only a few parameters. This goes hand in hand with analyzing also the structure of both an optimal and an algorithm's schedule. Then we express the total cost of both schedules as functions of these few parameters. It is noteworthy that, under the assumptions made, we typically characterize the *exact* worst-case ratios of the considered algorithms. Given the parameterized cost ratio, we analyze the worst-case parameter choice. This technical part involves second order analysis which we perform with computer assistance. These computations are provided as notebook- and pdf-files at a companion webpage.[1]

Two variants of the problem attracted our attention in particular. In an *uniform* instance all jobs have the same upper limit $\bar{p}_j$, which makes them initially undistinguishable to the scheduler. This means that the algorithm's decision whether to test a job, does not depend on the job itself, but only on the outcome of previous tests. Moreover in an *extreme uniform instance*, after testing a job j its processing time is either 0 or $\bar{p}_j$. This means that the benefit of a test is either maximized or none at all. Intuitively one would think that extreme uniform instances capture the worst case instances of the problem, hence it is not surprising that our lower bound constructions are of this form. In addition we design specific algorithms for these variants. One motivation was to follow a detour in order to find a better deterministic algorithm for the general problem, which unfortunately failed. Another motivation is that there is a huge amount of literature for scheduling problems with equal processing time. Therefore we believe that these variants are interesting for their own.

Our results are the following. For scheduling with testing on a single machine with the objective of minimizing the sum of completion times, we present a 2-competitive deterministic algorithm and prove that no deterministic algorithm can achieve competitive ratio less than 1.8546. We then present a 1.7453-competitive randomized algorithm, showing that randomization provably helps for this problem. We also give a lower bound of 1.626 on the best possible competitive ratio of any randomized algorithm. Both lower bounds hold even for *extreme uniform instances*, i.e. instances with uniform upper limits where every processing time is either 0 or equal to the upper limit.

[1] Files that can be opened with the algebraic solver *Mathematica* are available at the URL http://cslog.uni-bremen.de/nmegow/public/mathematica-SwT.zip.

For such instances we give a 1.8668-competitive algorithm. In the special case where the upper limit of all jobs is ≈ 1.9896, the value used in our deterministic lower bound construction, that algorithm is even 1.8552-competitive, which is nearly optimal. For the case of uniform upper limits and arbitrary processing times, we give a deterministic 1.9338-competitive algorithm. An overview of these results is shown in Table 1.

Finally, we give tight results for the simpler problem of minimizing the makespan in scheduling with testing. The best possible deterministic algorithm has competitive ratio $\varphi \approx 1.618$, where φ is the Golden ratio. The optimal randomized algorithm has competitive ratio $4/3$.

In the problem that we introduce in this paper, the interplay between the online algorithm and the adversary has a novel flavor due to the presence of tests: Testing a job forces the adversary to select a specific processing time right away, while otherwise the adversary can make this choice *after* the algorithm has completed all jobs. To our knowledge, this kind of interaction does not appear in the standard online computation framework.

From a technical perspective, our contribution consists of two parts. First we present techniques to modify instances in an adversarial manner, while reducing the number of distinct job parameters. This allows us to describe the competitive ratio with a few parameters. Second we show how second order analysis can be used to optimize these parameters.

Organization of the paper. In Section 2, we give the problem definition, some observations and structural properties. Section 3 is devoted to lower and upper bounds for deterministic algorithms for general instances for minimizing the sum of completion times. Section 4 addresses randomized algorithms. In Section 5, we give more fine-grained results for special cases of the problem with uniform upper bounds. Finally, in Section 6 we give optimal deterministic and randomized algorithms for minimizing the makespan.

1.3 Related work

The arguably most classical framework modeling sequential decision making problems with an exploration-exploitation trade-off is the stochastic multi-armed bandit problem. In each round, one choses from a set of actions (bandit arms) and obtains some observable payoff, where the goal is to maximize the total payoff. Since its introduction in 1933 in [57] a plethora of variants has been analyzed and till today this is an actively studied area with applications particularly in online auctions, adaptive routing, and the theory of learning in games; see e.g. [9, 24].

One of the oldest stochastic problems with explicit exploration cost is Weitzman's Pandora's box problem [58]. Given n random variables with probability distributions, the goal is to find a single variable of largest value, but one needs to pay a cost for each probe of a variable. Its solution can be stated as a special case of the Gittins index theorem [25, 36]. A nice exposition of an application of a variation of the Gittins index to a problem that can be stated as 'playing golf with two balls' can be found in [15]. Only recently, combinatorial otimization problems have been studied in this context with the goal of optimizing the sum of query costs and the objective value of the selected solution. This includes problems such as matching, set cover, facility

location, and prize-collecting Steiner tree; see, e.g., [27, 56] and references therein, also with uncertainty in the cost function [41].

Other stochastic problems taking exploration cost into account, such as stochastic knapsack [13, 40], orienteering [5, 28], matching [3, 4, 6, 7, 11], and probing problems [1, 29, 30], employ a *query-commit* model, which means that queried elements must be part of the solution, or it is required that the solution elements are queried. These are quite strong restrictions which change the nature of the benefit-cost trade-off that an algorithm experiences when making queries.

All these models have in common that the uncertain information follows some stochastic model. We follow a more pessimistic approach, by studying an online or robustness model where the algorithm has no prior stochastic information. As usual in the absence of a known distribution, we assume the worst case and let an adversary chose the hidden information.

This adversarial model falls in the area of deterministic explorable (or queryable) uncertainty, where additional information about the input can be learned using a query operation, a test in our setting. The line of research on optimization with explorable uncertain data has been initiated by Kahan [32] in 1991. His work concerns selection problems with the goal of minimizing the number of queries that are necessary to find the optimal solution. After the initiation by Kahan [32] on selection problems. further problems have been studied in this uncertainty model including finding the k-th smallest value in a set of uncertainty intervals [19, 31, 32] (also with non-uniform query cost [19]), caching problems in distributed databases [50], computing a function value [35], and classical combinatorial optimization problems, such as shortest path [18], finding the median [19], the knapsack problem [26], and the MST problem [17, 23, 43]. While most work aims for minimal query sets to guarantee exact optimal solutions, Olsten and Widom [50] initiate the study of trade-offs between the number of queries and the precision of the found solution. They are concerned with caching problems. Further work in this vein can be found in [18, 19, 35].

In all this previous work, the execution of queries is separate from the actual optimization problem being solved. In our case, the tests are executed by the same machine that runs the jobs. Hence, the tests are not considered separately, but they directly affect the objective value of the actual problem (by delaying the completion of other jobs while a job is being tested). Therefore, instead of minimizing the number of tests needed until an optimal schedule can be computed (which would correspond to the standard approach in the work on explorable uncertainty discussed above), in our case the tests of jobs are part of the schedule, and we are interested in the sum of completion times as the single objective function.

Our adversarial model is inspired by (and draws motivation from) recent work on a stochastic model of scheduling with testing introduced by Levi, Magnanti and Shaposhnik [39, 55]. They consider the problem of minimizing the weighted sum of completion times on one machine for jobs whose processing times and weights are random variables with a joint distribution, and are independent and identically distributed across jobs. In their model, testing a job does not make its processing time shorter, it only provides information for the scheduler (by revealing the exact weight and processing time for a job, whereas initially only the distribution is known). They present structural results about optimal policies and efficient optimal or near-optimal solutions based on dynamic programming.

Scheduling problems, in general, have been studied extensively over decades. They occur in many different variations in a wide range of applications ranging from tradi-

tional production scheduling and project planning to new resource management tasks arising in the advent of internet technology such as distributed cloud service networks and the allocation or virtual machines to physical servers. For a general overview and classification, we refer to the reference works [38, 52].

The most common frameworks for modeling scheduling with uncertain input are stochastic scheduling [45, 47, 48], online scheduling [22, 53], a generalization of the former two [12, 44] and robust scheduling [14, 33, 37]. These models differ in the way that information is made available to an algorithm and in the performance metrics. We do not aim at a comprehensive review and, instead, refer the reader to the pointers in the literature. Regarding the access to information, our scheduling with testing model is closest to online and robust optimization, where information (e.g. about job processing times) is revealed incrementally and adversarially. However, in stochastic scheduling, a job's processing time can be explored by partially executing a job and observing its processing time. Clearly, there is much less flexibility in exploiting the learned information, than when testing, as the job might have finished before any action can be taken.

A new learning-based scheduling model was proposed by Marban, Rutten and Vredeveld [42]. They introduce a Bayesian model, in which jobs belong to classes and the stochastic processing times of jobs in the same class are drawn from the same unknown distribution. This distribution can be learnt by executing jobs. Besides this Bayesian model and the aforementioned stochastic model of scheduling with testing by Levi et al. [39], none of the traditional uncertainty models for scheduling takes the opportunity of actively exploring unknown information at some cost into account explicitly.

Finally, it appears noteworthy that the concept of taking exploration cost into account when dealing with uncertainty gains momentum also in other fields such as, e.g., random graphs. Recently some research papers ask the question of how many edges must be queried in a given random graph, in order to verify that some graph property is satisfied. There are results on finding Hamiltonian cycles [20] and finding paths [21] in random graphs with few queries.

2 Preliminaries

Problem definition. The problem of scheduling with testing is defined as follows. We are given n jobs to be scheduled on a single machine. Each job j has an upper limit on the processing time[2] $\bar{p}_j \in \mathbb{Q}^+$. It can either be executed untested (taking time $\bar{p}_j$), or be tested (taking time 1) and then executed at an arbitrary later time (taking time $p_j \in \mathbb{Q}^+$, where $0 \leq p_j \leq \bar{p}_j$). Initially only $\bar{p}_j$ is known for each job, and p_j is only revealed after j is tested. The machine can either test or execute a job at any time. The completion time of job j is denoted by C_j. Unless noted otherwise, we consider the objective of minimizing the sum of completion times $\sum_j C_j$.

The optimal offline solution. If the processing times p_j that jobs have after testing are known, an optimal schedule is easy to determine: Testing and executing job j takes time $1 + p_j$, so it is beneficial to test the job only if $1 + p_j < \bar{p}_j$. Since the SPT rule is optimal for minimizing the sum of completion times, in the optimal schedule, jobs are

[2] We define the problem with rational numbers for the ease of representing them in a computer, but all our results and proofs also hold for real numbers.

ordered by non-decreasing $\min\{1+p_j, \bar{p}_j\}$. In this order, the jobs with $1+p_j < \bar{p}_j$ are tested and executed while jobs with $1+p_j \geq \bar{p}_j$ are executed untested. (For jobs with $1+p_j = \bar{p}_j$ it does not matter how they are processed.)

Performance analysis. We compare the performance of an algorithm ALG to the optimal schedule using competitive analysis [8]. We denote by $\text{ALG}(I)$ the objective value (cost) of the schedule produced by ALG for an instance I, and by $\text{OPT}(I)$ the optimal cost. An algorithm ALG is *ρ-competitive* or *has competitive ratio at most ρ* if $\text{ALG}(I)/\text{OPT}(I) \leq \rho$ for all instances I of the problem. For randomized algorithms, $\text{ALG}(I)$ is replaced by $E[\text{ALG}(I)]$ in this definition. If the instance I is clear from the context and no confusion can arise, we also write ALG for $\text{ALG}(I)$ and OPT for $\text{OPT}(I)$.

When we analyze an algorithm or the optimal schedule, we will typically first argue that the schedule has a certain structure with different blocks of tests or job completions. Once we have established that structure, the cost of the schedule can be calculated by adding the cost for each block taken in isolation, plus the effect of the block on the completion times of later jobs. For example, assume that we have n jobs with upper limit $\bar{p}$, that αn of these jobs are *short*, with processing time 0, and $(1-\alpha)n$ jobs are *long*, with processing time $\bar{p}$. If an algorithm (in the worst case) first tests the $(1-\alpha)n$ long jobs, then tests the αn short jobs and executes them immediately, and finally executes the $(1-\alpha)n$ long jobs that were tested earlier (see also Figure 1), the total cost of the schedule can be calculated as

$$(1-\alpha)n^2 + \frac{\alpha n(\alpha n+1)}{2} + \alpha n(1-\alpha)n + \frac{(1-\alpha)n((1-\alpha)n+1)}{2}\bar{p}$$

where $(1-\alpha)n^2$ is the total delay that the $(1-\alpha)n$ tests of long jobs add to the completion times of all n jobs, $\frac{\alpha n(\alpha n+1)}{2}$ is the sum of completion times of a block with αn short jobs that are tested and executed, $\alpha n(1-\alpha)n$ is the total delay that the block of short jobs with total length αn adds to the completion times of the $(1-\alpha)n$ jobs that come after it, and $\frac{(1-\alpha)n((1-\alpha)n+1)}{2}\bar{p}$ is the sum of completion times for a block with $(1-\alpha)n$ job executions with processing time $\bar{p}$ per job.

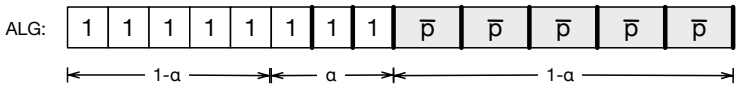

Fig. 1 Typical schedule produced by an algorithm. White boxes represent tests and grey boxes represent job executions. The completion time of a job is depicted by a thick bar. Test and execution of a job might be separated. A job of length 0 completes immediately after its test.

Lower limits. A natural generalization of the problem would be to allow each job j to have, in addition to its upper limit $\bar{p}_j$, also a lower limit ℓ_j, such that the processing time after testing satisfies $\ell_j \leq p_j \leq \bar{p}_j$. We observe that the presence of lower limits has no effect on the optimal schedule, and can only help an algorithm. As we are interested in worst-case analysis, we assume in the remainder of the paper that every job has a lower limit of 0. Any algorithm that is ρ-competitive in this case is also ρ-competitive in the case with arbitrary lower limits (the algorithm can simply ignore the lower limits).

Preemption. The ability to preempt the execution of a test or of a (tested or untested) job would be of no benefit to the algorithm or the adversary as no new information is obtained during the execution. Therefore, we only consider algorithms and schedules that do not preempt tests and that do not preempt job executions. However, as noted above, the execution of a tested job can be scheduled any time after the completion of the test.

Jobs with small $\bar{p}_j$. We will consider several algorithms and prove competitiveness for them. We observe that any ρ-competitive algorithm may process jobs with $\bar{p}_j < \rho$ without testing in order of increasing $\bar{p}_j$ at the beginning of its schedule.

Lemma 1 *Without loss of generality any algorithm* Alg *(deterministic or randomized) claiming competitive ratio ρ starts by scheduling untested all jobs j with $\bar{p}_j < \rho$ in increasing order of $\bar{p}_j$. Moreover, worst case instances for* Alg *consist solely of jobs j with $\bar{p}_j \geq \rho$.*

Proof We transform Alg into an algorithm Alg$'$ which obeys the claimed behavior and show that its ratio does not exceed ρ. Consider an arbitrary instance I.

Let J be the sequence of jobs j with $\bar{p}_j < \rho$ ordered in increasing $\bar{p}_j$ order. We divide J into J_0, J_1, where J_0 consists of the jobs j with $0 \leq \bar{p}_j < 1$ and J_1 consists of the jobs j with $1 \leq \bar{p}_j < \rho$. Alg$'$ starts by executing the job sequence J untested, and then schedules all remaining jobs as Alg, following the same decisions to test and the order of tests and executions. In a worst-case instance all the jobs in J have processing time 0. By optimality of the SPT policy Opt schedules first J_0 untested as well, and then schedules J_1 tested spending time 1 on each job. The ratio of the costs of these parts is

$$\frac{\textsc{Alg}'(J)}{\textsc{Opt}(J)} < \rho$$

where the inequality follows from $\bar{p}_j / \min\{1, \bar{p}_j\} < \rho$ for all $j \in J$. Let len denote the length of a schedule. Then by the same argument we have

$$\frac{\text{len}(\textsc{Alg}'(J))}{\text{len}(\textsc{Opt}(J))} < \rho.$$

Let I' be the instance I without the jobs in J. Let k be the number of jobs in I'. Since I' contains only jobs with large upper limit, we have $\textsc{Alg}(I') = \textsc{Alg}'(I')$. We have

$$\begin{aligned}\textsc{Alg}'(I) &= \textsc{Alg}'(J) + k \cdot \text{len}(\textsc{Alg}'(J)) + \textsc{Alg}'(I') \\ \textsc{Opt}(I) &= \textsc{Opt}(J) + k \cdot \text{len}(\textsc{Opt}(J)) + \textsc{Opt}(I').\end{aligned}$$

From these (in)equalities we conclude

$$\begin{aligned}\frac{\textsc{Alg}(I)}{\textsc{Opt}(I)} \leq \rho &\Rightarrow \frac{\textsc{Alg}'(I)}{\textsc{Opt}(I)} \leq \rho \\ \frac{\textsc{Alg}(I)}{\textsc{Opt}(I)} \geq \rho &\Rightarrow \frac{\textsc{Alg}'(I)}{\textsc{Opt}(I)} \leq \frac{\textsc{Alg}(I')}{\textsc{Opt}(I')}\end{aligned}$$

which means that if Alg is ρ competitive then so is Alg$'$ and that there are worst-case instances for Alg only with jobs having upper limit at least ρ. □

Increasing or decreasing ALG *and* OPT. Throughout the paper we sometimes consider worst-case instances consisting of only a few different job types. In order to do so we need to change carefully the parameters of a given instance, in such a way that the competitive ratio does not decrease and the number of distinct job types decreases. The following generic proposition allows us to do so in some cases.

Proposition 2 *Fix some algorithm* ALG *and consider a family of instances described by some parameter $x \in [\ell, u]$, which could represent p_j or $\bar{p}_j$ for some job j or for some set of jobs. Suppose that both* OPT *and* ALG *are linear in x for the range $[\ell, u]$. Then the ratio* ALG/OPT *is maximized, among all choices of $x \in [\ell, u]$, for at least one of the two choices $x = \ell$ or $x = u$. Moreover, if* OPT *and* ALG *are increasing in x with the same slope, then this holds for $x = \ell$.*

Proof The proof follows from the fact that an expression of the form $\text{ALG}/\text{OPT} = (a + bx)/(a' + b'x)$ is monotone in x. Indeed its derivative is

$$\frac{a'b - ab'}{(a' + b'x)^2}$$

whose sign does not depend on x. The last statement follows from the fact that if $\text{ALG} > \text{OPT}$ and $0 < \delta \leq \text{OPT}$ then $(\text{ALG} - \delta)/(\text{OPT} - \delta) > \text{ALG}/\text{OPT}$. □

We can make successive use of this proposition in order to show useful properties on worst-case instances.

Lemma 3 *Suppose that there is an interval $[\ell', u']$ such that* OPT *schedules all jobs j with $p_j \in [\ell', u']$ either all tested or all untested, independently of the actual processing time in $[\ell', u']$. Suppose that this holds also for* ALG*. Moreover, suppose that the schedules of* OPT *and* ALG *do not change (in the sense that the order of all tests and job executions remains the same) when changing the processing times in $[\ell', u']$ as long as the relative ordering of job processing times does not change. Then there is a worst-case instance for* ALG *where every job j with $p_j \in [\ell', u']$ satisfies $p_j \in \{\ell', u'\}$.*

Proof Fix some worst-case instance for the algorithm ALG. Let S be the set of jobs j with $p_j = x$ for some x with $\ell' < x < u'$. Let ℓ, u be the values $\ell = \max(\{\ell'\} \cup \{p_i : p_i < x\})$ and $u = \min(\{u'\} \cup \{p_i : p_i > x\})$. Informally ℓ is the largest processing time strictly smaller than x or ℓ' if x is already the smallest processing time or if this would make ℓ smaller than ℓ'. Also u is the smallest processing time strictly larger than x or u' if x is already the largest processing time or if this would exceed u'. Since the schedules are preserved when changing the processing times of S, both costs ALG and OPT are linear in x within $[\ell, u]$. Now we can use Proposition 2 to show that there is a worst-case instance where all jobs in S have processing time either ℓ or u. In both cases we have reduced the number of distinct processing times strictly being between ℓ' and u'. By repeating this argument sufficiently often we obtain the claimed statement. □

3 Deterministic Algorithms

3.1 Algorithm THRESHOLD

We show a competitive ratio of 2 for a natural algorithm that uses a threshold to decide whether to test a job or execute it untested.

Algorithm 1 (THRESHOLD) *First jobs with $\bar{p}_j < 2$ are scheduled in order of non-decreasing upper limits without testing. Then all remaining jobs are tested. If the revealed processing time of job j is $p_j \leq 2$ (short jobs), then the job is executed immediately after its test. After all pending jobs (long jobs) have been tested, they are scheduled in order of increasing processing time p_j.*

By Lemma 1 we may restrict our competitive analysis w.l.o.g. to instances with $\bar{p}_j \geq 2$. Note, that on such instances THRESHOLD tests all jobs. From a simple interchange argument it follows that the structure of the algorithm's solution in a worst-case instance is as follows.

- Test phase: The algorithm tests all jobs that have $p_j > 2$, and defers them.
- Short jobs phase: The algorithm tests short jobs ($p_j \leq 2$) and executes each of them right away. The jobs are tested in order of non-increasing processing time.
- Long jobs phase: The algorithm executes all deferred long jobs in order of non-decreasing processing times.

An optimal solution will not test jobs with $p_j + 1 \geq \bar{p}_j$. It sorts jobs in non-decreasing order of values $\min\{1 + p_j, \bar{p}_j\}$.

First, we analyze and simplify worst-case instances.

Lemma 4 *There is a worst-case instance for* THRESHOLD *in which all short jobs with $p_j \leq 2$ have processing time either 0 or 2.*

We give a proof without modifying upper limits, which is not necessary in this section but will come handy later when we analyze THRESHOLD for arbitrary uniform upper limits.

Proof Consider short jobs that are tested by both, the optimum and THRESHOLD, i.e., short jobs with $p_j < \bar{p}_j - 1$. We argue that we can either decrease the processing time of a short job j to 0 or increase it to $\min\{2, \bar{p}_j - 1\}$ without decreasing the worst-case ratio. With respect to the order in which THRESHOLD executes the jobs, let ℓ be the first short job with $p_\ell < \min\{2, \bar{p}_\ell - 1\}$ and let i be the last short job with $p_i > 0$.

Suppose $i \neq \ell$. Let $\Delta = \min\{p_i, \min\{2, \bar{p}_\ell - 1\} - p_\ell\}$. We decrease p_i by Δ and at the same time increase p_ℓ by Δ. The value Δ is chosen in such a way that either p_i will become 0 or p_ℓ will be $\min\{2, \bar{p}_\ell - 1\}$, as desired. The schedule produced by the algorithm will be the same except that jobs $\ell, \ldots, i-1$ complete Δ units later. In the optimal schedule ℓ and i are scheduled in opposite order. Suppose we keep the schedule fixed when changing the processing times of jobs i and ℓ. Then i's completion time as well as those of jobs between i and ℓ decreases. In an optimal schedule jobs might be re-ordered, but this only improves the total objective further. Hence, the total ratio of objective values does not decrease.

Now, assume $i = \ell$, i.e., there is exactly one short job with processing time p_i strictly between 0 and $\min\{2, \bar{p}_i - 1\}$. We argue that either increasing or decreasing p_i to $\min\{2, \bar{p}_i - 1\}$ or 0 will not decrease the worst-case ratio. Such a change Δ does not change the order of jobs in the algorithm's solution and thus the change in the objective is Δ times the number of jobs completing after i. In an optimum solution, there are untested short or long jobs which are scheduled between short tested jobs and their relative order with i may change when i is in-/decreased by Δ. However, let us consider a possibly not optimal schedule that simply does not adjust the order after changing i. Then the change in the objective is linear in Δ in the above-given range,

as it is for the algorithm, and thus, by Proposition 2 either increasing or decreasing p_i by Δ does not decrease the ratio of objective values. Now, the truly optimal objective value is not larger and thus, the true worst-case ratio is not smaller.

Now, we may assume that all short jobs remaining with processing times different from 0 and 2 are untested in the optimum solution because their processing time is at least $\bar{p}_j - 1$. Again, the optimum does not test those jobs, and hence, increasing the processing time to 2 has no impact on the optimal schedule, while our algorithm's cost only increases. Thus, the worst-case ratio increases, which concludes the proof. □

Threshold tests all jobs and makes scheduling decisions depending on job processing times p_j but independently of upper limits of jobs. Since all short jobs have $p_j \in \{0, 2\}$, we can reduce all their upper limits to $\bar{p}_j = 2$ without affecting the schedule, whereas it may only improve the optimal schedule. In particular we may assume now the following.

Lemma 5 *There is a worst-case instance in which all short jobs have $\bar{p}_j = 2$ and execution times are 0 or 2.*

Lemma 6 *There is a worst-case instance in which long jobs with $p_j > 2$ have a uniform upper limit $\bar{p}$ and processing times $p_j = \bar{p}_j = 2 + \epsilon$ for infinitesimally small $\epsilon > 0$.*

Proof For all long jobs, which are tested by the optimum, we reduce the upper limit to $\bar{p}_j = 1 + p_j$. This does not change the algorithm's solution. But the optimum may as well run those previously tested jobs also untested and would not change its total objective value.

Now the optimum solution runs all long jobs without testing them. Thus, increasing the processing time of long jobs to $p_j = \bar{p}_j$ does not affect the optimum cost whereas the algorithm's cost increases.

Lemma 5 implies that all long jobs are scheduled in the same order by the algorithm and an optimum without any short jobs in between. Then, setting $\bar{p} = 2 + \epsilon$ decreases the objective values of both algorithms by the same amount and thus does not decrease the ratio. The lemma follows. □

Now we are ready to prove the main result.

Theorem 7 *Algorithm* Threshold *has competitive ratio at most 2 for scheduling with testing with the objective of minimizing the sum of completion times.*

Proof We consider worst-case instances of the type derived above. Let a be the number of short jobs with $p_j = 0$, let b be the number of short jobs with $\bar{p}_j = p_j = 2$, and let c be the number of long jobs with $\bar{p}_j = 2 + \epsilon$, see Figure 2.

Threshold's solution for a worst-case instance first tests all long jobs, then tests and executes the short jobs in decreasing order of processing times, and completes with the executions of long jobs. The total objective value ALG is

$$\textsc{Alg} = (a+b+c)c + b(b+1)/2 \cdot 3 + 3b(a+c) + \\ + a(a+1)/2 + a \cdot c + c(c+1)/2 \cdot (2+\epsilon).$$

An optimum solution tests and schedules first all 0-length jobs and then executes the remaining jobs without tests. The objective value is

$$\textsc{Opt} = a(a+1)/2 + a(b+c) + b(b+1)/2 \cdot 2 + 2bc + c(c+1)/2 \cdot (2+\epsilon).$$

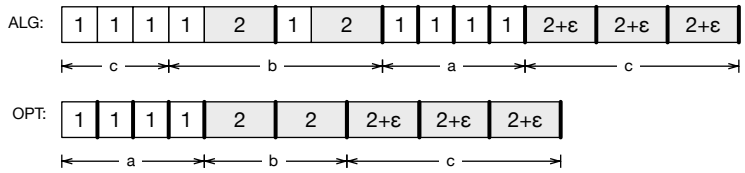

Fig. 2 Worst case instance for Threshold.

Simple transformation shows that $\textsc{Alg} \leq 2 \cdot \textsc{Opt}$ is equivalent to

$$\begin{aligned} 2ab + 2c^2 \leq a^2 + b^2 + a + b + c(c+1)(2+\epsilon) \quad &\Leftrightarrow \\ 0 \leq (a-b)^2 + a + b + c^2\epsilon + c(2+\epsilon), \end{aligned}$$

which is obviously satisfied and the theorem follows. □

Note that the analysis of Threshold is tight. Indeed it has ratio $2-\varepsilon$ on the instance consisting of a single job j with $p_j = 0$ and $\bar{p}_j = 2-\varepsilon$, for arbitrarily small $\varepsilon > 0$. The algorithm does not test the job, but the optimal schedule does.

We conclude this section with an observation that was brought to our attention. Consider a slight modification of Threshold that delays *all* jobs after their test, regardless of the revealed processing time. This algorithm, which we call DelayAll, seems to produce worse schedules than Threshold. For example, when all jobs have upper bound 2 and processing time 0, DelayAll has a cost of n^2, while Threshold has a cost of $n(n+1)/2$. Nevertheless, the competitive ratio of DelayAll is also 2, which can be shown as follows: Again, by Lemma 1 we may assume that all jobs have upper limit at least 2. Hence, DelayAll starts by testing all jobs, and then executes them in order of non-decreasing processing times. By Lemma 3, we can assume that all jobs j with $0 \leq p_j \leq 1$ (which are tested by both Opt and DelayAll) satisfy $p_j \in \{0,1\}$. For the jobs with $p_j \in [1,2]$, we can then first set $\bar{p}_j = 2$ (this can only help Opt) and next set $p_j = 2$ (this can only increase the cost of DelayAll but does not affect Opt). Finally, we can set $\bar{p}_j = p_j$ for all jobs with $p_j > 2$ (this can only help Opt) and then set $p_j = \bar{p}_j = 2$ for all these jobs (this decreases the objective values of Opt and DelayAll by the same amount and thus does not decrease the ratio). Let the resulting instance consist of a jobs with processing time 0 and b jobs with processing time 2. The cost of DelayAll is $A = (a+b)(a+b) + b(b+1) = a^2 + 2ab + 2b^2 + b$, and the cost of Opt is $\frac{1}{2}a(a+1) + ab + b(b+1) = \frac{a^2}{2} + \frac{a}{2} + ab + b^2 + b \geq \frac{1}{2}A$.

3.2 Deterministic lower bound

In this section we give a lower bound on the competitive ratio of any deterministic algorithm. The instances constructed by the adversary have a very special form: All jobs have the same upper limit $\bar{p}$, and the processing time of every job is either 0 or $\bar{p}$.

Consider instances of n jobs with uniform upper limit $\bar{p} > 1$, and consider any deterministic algorithm. We say that the algorithm *touches* a job when it either tests the job or executes it untested. We re-index jobs in the order in which they are first touched by the algorithm, i.e., job 1 is the first job touched by the algorithm and job

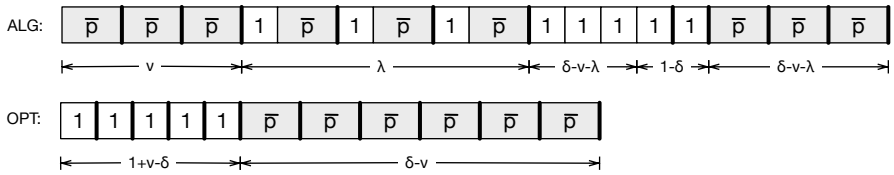

Fig. 3 Lower bound construction.

n is the last. The adversary fixes a fraction $\delta \in [0,1]$ and sets the processing time of job j, $1 \le j \le n$, to:

$$p_j = \begin{cases} 0 & \text{, if } j \text{ is executed by the algorithm untested, or } j > \delta n \\ \bar{p} & \text{, if } j \text{ is tested by the algorithm and } j \le \delta n \end{cases}$$

A job j is called *short* if $p_j = 0$ and *long* if $p_j = \bar{p}$. Let j_0 be the smallest integer that is greater than δn. Job j_0 is the first of the last $(1-\delta)n$ jobs that are short no matter whether the algorithm tests them or not.

We assume the algorithm knows $\bar{p}$ and δ, which can only improve the performance of the best-possible deterministic algorithm. Note that with δ and $\bar{p}$ known to the algorithm, it has full information about the actions of the adversary. Nevertheless, it is still non-trivial for an algorithm to decide for each of the first δn jobs whether to test it (which makes the job a long job, and hence the algorithm spends time $\bar{p}+1$ on it while the optimum executes it untested and spends only time $\bar{p}$) or to execute it untested (which makes it a short job, and hence the algorithm spends time $\bar{p}$ on it while the optimum spends only time 1).

Let us first determine the structure of the schedule produced by an algorithm that achieves the best possible competitive ratio for instances created by this adversary, as displayed in Figure 3.

Lemma 8 *The schedule of a deterministic algorithm with best possible competitive ratio has the following form, where $\lambda, \nu \ge 0$ and $\nu + \lambda \le \delta$: The algorithm first executes νn jobs untested, then tests and executes λn long jobs, then tests $(\delta - \nu - \lambda)n$ long jobs and delays their execution, then tests and executes the remaining $(1-\delta)n$ short jobs, and finally executes the $(\delta - \nu - \lambda)n$ delayed long jobs that were tested earlier, see Figure 3.*

Proof It is clear that the algorithm will test the last $(1-\delta)n$ jobs and execute each such job (with processing time 0) right after its test, as executing any of them untested does not affect the optimal solution but increases the objective value of the algorithm. Furthermore, consider the time t when the algorithm tests job j_0. From this time until the end of the schedule, the algorithm will test and execute the last $(1-\delta)n$ jobs (spending time 1 on each such job), and execute all the long jobs that were tested earlier but not yet executed (spending time $\bar{p} > 1$ on each such job). As the SPT rule is optimal for minimizing the sum of completion times, it is clear that from time t onward the algorithm will first test and execute the $(1-\delta)n$ short jobs and afterwards execute the long jobs that were tested but not executed before time t.

Before time t, the algorithm touches the first δn jobs. Each of these can be executed untested (let νn be the number of such jobs), or tested and also executed before time t

(let λn be the number of such jobs), or tested but not executed before time t (this happens for the remaining $(\delta-\nu-\lambda)n$ jobs). To minimize the sum of completion times of these jobs, it is clear that the algorithm first executes the νn jobs untested (spending time $\bar{p}$ per job), then tests the λn long jobs and executes each of them right after its test (spending time $1+\bar{p}$ per job), and finally tests the remaining $(\delta-\nu-\lambda)n$ long jobs. □

The cost of the algorithm in dependence on ν, λ, δ and $\bar{p}$ can now be expressed as:

$$\begin{aligned}\textsc{Alg}(\nu,\lambda,\delta,\bar{p}) &= n^2\Big(\frac{\nu^2}{2}\bar{p}+\nu\bar{p}(1-\nu)+\frac{\lambda^2}{2}(1+\bar{p})+\lambda(1+\bar{p})(1-\nu-\lambda)\\ &\quad+(\delta-\nu-\lambda)(1-\nu-\lambda)+\frac{(1-\delta)^2}{2}+(1-\delta)(\delta-\nu-\lambda)+\frac{(\delta-\nu-\lambda)^2}{2}\bar{p}\Big)+O(n)\\ &= \frac{n^2}{2}\Big(1+2\delta(1-\nu\bar{p})+\delta^2(\bar{p}-1)+2\nu(\nu+\bar{p}-2)+\lambda^2+2\lambda(\nu+\bar{p}-1-\delta\bar{p})\Big)+O(n)\end{aligned}$$

The optimal schedule first tests and executes the $(\nu+1-\delta)n$ short jobs and then executes the $(\delta-\nu)n$ long jobs untested. Hence, the optimal cost, which depends only on ν, δ and $\bar{p}$, is:

$$\begin{aligned}\textsc{Opt}(\nu,\delta,\bar{p}) &= n^2\left(\frac{(\nu+1-\delta)^2}{2}+(\nu+1-\delta)(\delta-\nu)+\frac{(\delta-\nu)^2}{2}\bar{p}\right)+O(n)\\ &= \frac{n^2}{2}\Big(1+(\delta-\nu)^2(\bar{p}-1)\Big)+O(n)\end{aligned}$$

We introduce the notations

$$\textsc{Alg}'(\nu,\lambda,\delta,\bar{p}) = \lim_{n\to\infty}\frac{2}{n^2}\textsc{Alg}(\nu,\lambda,\delta,\bar{p}) \qquad \text{and}$$
$$\textsc{Opt}'(\nu,\delta,\bar{p}) = \lim_{n\to\infty}\frac{2}{n^2}\textsc{Opt}(\nu,\delta,\bar{p}).$$

As the adversary can choose δ and $\bar{p}$, while the algorithm can choose ν and λ, the value

$$R = \max_{\delta,\bar{p}}\min_{\nu,\lambda}\frac{\textsc{Alg}'(\nu,\lambda,\delta,\bar{p})}{\textsc{Opt}'(\nu,\delta,\bar{p})}$$

gives a lower bound on the competitive ratio of any deterministic algorithm in the limit for $n\to\infty$. By making n sufficiently large, the adversary can create instances with finite n that give a lower bound arbitrarily close to R.

The exact optimization of δ and $\bar{p}$ is rather tedious and technical as it involves the optimization of rational functions of several variables. In the following, we therefore only show that the choices $\delta=0.6306655$ and $\bar{p}=1.9896202$ give a lower bound of 1.854628 on the competitive ratio of any deterministic algorithm. (The fully optimized value of R is less than $1.1\cdot10^{-7}$ larger than this value.) For this choice of δ and $\bar{p}$ we have:

$$\begin{aligned}\textsc{Alg}'(\nu,\lambda,\delta,\bar{p}) &\approx 1.32747+\nu(\nu-1.26516)+\frac{1}{2}\lambda^2+\lambda(\nu-0.265165)\\ \textsc{Opt}'(\nu,\delta,\bar{p}) &\approx 0.696805+\nu(0.49481\nu-0.624119)\end{aligned}$$

The part of $\textsc{Alg}'(\nu,\lambda,\delta,\bar{p})$ involving λ is $\frac{1}{2}\lambda^2+\lambda(\nu+\bar{p}-1-\delta\bar{p})$, which is a quadratic function minimized at $\lambda=1+\delta\bar{p}-\bar{p}-\nu\approx0.265165-\nu$. As λ must be non-negative, we distinguish two cases depending on whether this expression is non-negative or not. Let $\tau=1+\delta\bar{p}-\bar{p}\approx0.265165$.

Case 1: $\nu \leq \tau$. In this case the best choice of λ for the algorithm is $\lambda = \tau - \nu$. The ratio $\textsc{Alg}'/\textsc{Opt}'$ then simplifies to:

$$f(\nu) = \frac{1.29231 + \nu(\frac{1}{2}\nu - 1)}{0.696805 + \nu(0.49481\nu - 0.624119)} = 1.01049 + \frac{1.18874 - 0.746417\nu}{1.40823 + \nu(\nu - 1.26133)}$$

In the range $0 \leq \nu \leq \tau$, the only local extremum of this function is a local maximum at $\nu \approx 0.201266$, so the function attains its minimum in the range at one of the two endpoints. As we have $f(\tau) > f(0) \approx 1.854628$, the function is minimized at $\nu = 0$, giving a lower bound of 1.854628 on the competitive ratio.

Case 2: $\nu > \tau$. In this case, the best choice of λ for the algorithm is $\lambda = 0$. The ratio $\textsc{Alg}'/\textsc{Opt}'$ then becomes:

$$g(\nu) = \frac{1.32747 + \nu(\nu - 1.26516)}{0.696805 + \nu(0.49481\nu - 0.624119)} = 2.02098 + \frac{-0.163208 - 0.00774781\nu}{1.40823 + \nu(\nu - 1.26133)}.$$

This function is monotonically decreasing in the range $\tau < \nu \leq \delta$, so it is minimized for $\nu = \delta$, giving a ratio of $g(\delta) \approx 1.854628$.

As we get a lower bound of 1.854628 in both cases, this lower bound holds generally.

Theorem 9 *No deterministic algorithm can achieve a competitive ratio or asymptotic competitive ratio below* 1.854628 *for scheduling with testing with the objective of minimizing the sum of completion times. This holds even for instances with uniform upper limit where each processing time is either* 0 *or equal to the upper limit.*

4 Randomized Algorithms

4.1 Algorithm Random

Algorithm 2 (Random) *The randomized algorithm* Random *has parameters* $1 \leq T \leq E$ *and works in 3 phases. First it executes all jobs with* $\bar{p}_j < T$ *without testing in order of increasing* $\bar{p}_j$*. Then it tests all jobs with* $\bar{p}_j \geq T$ *in uniform random order. Each tested job* j *is executed immediately after its test if* $p_j \leq E$ *and is deferred otherwise. Finally all deferred jobs are executed in order of increasing processing time.*

We analyze the competitive ratio of Random, and optimize the parameters T, E such that the resulting competitive ratio is T.

By Lemma 1 we restrict to instances with $\bar{p}_j \geq T$ for all jobs. Then, the schedule produced by Random can be divided into two parts. Part (1) contains all tests, of which those that yield processing time p_j at most E are immediately followed by the job's execution. Part (2) contains all jobs that *have been* tested and with processing time larger than E. These jobs are ordered by increasing processing time. Jobs in the first part are completed in an arbitrary order.

Furthermore, we can assume $\bar{p}_j = \max\{p_j, T\}$ for all jobs. Reducing $\bar{p}_j$ to this value does not change the cost or behavior of Random, but may decrease the cost of Opt. We make further assumptions along the following lines. Let $\epsilon > 0$ be an arbitrary small number such that $p_j \geq E + \epsilon$ for all jobs j with $p_j > E$. These jobs are executed by Random in part (2) of the schedule in non-decreasing order of processing time. The same holds for Opt, which by the SPT *Policy* also schedules these jobs in the end in

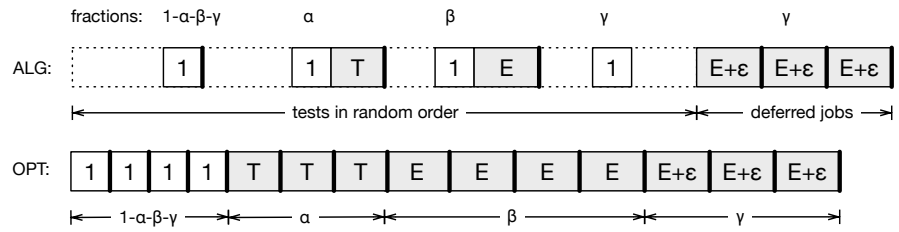

Fig. 4 Worst case analysis of the algorithm Random.

exactly the same order. Hence if we set $\bar{p}_j = p_j = E + \epsilon$ for all these jobs, then we reduce the objective value of Random and of Opt by the same value. According to Proposition 2 this transformation only increases the competitive ratio of the algorithm.

Using again the assumption that $\bar{p}_j = \max\{p_j, T\}$ for all jobs, we now have that all jobs j in part (2) satisfy $\bar{p}_j = p_j = E + \epsilon$ and the remaining jobs satisfy either $\bar{p}_j = p_j \in [T, E]$ or $\bar{p}_j = T$ and $p_j \leq T$. Now we apply Lemma 3 to show that for all jobs j with $\bar{p}_j = p_j \in [T, E]$ we can in fact assume $\bar{p}_j = p_j \in \{T, E\}$. The usage of the lemma is a bit subtle as the output of Random is a distribution of schedules. For any fixed scheduling order corresponding to a realization of the random execution of the algorithm, the conditions of the lemma are satisfied. But we cannot apply the lemma on each order individually, as we might end up with different problem instances. However, the expected cost of Random is linear in the execution times of jobs j within $p_j \in [T, E]$. This is the key condition which is used in the proof of Lemma 3. Hence we conclude that the statement of the lemma still holds.

Now we turn to jobs j with $\bar{p}_j = T$ and $p_j \leq T$. For the jobs with $0 \leq p_j \leq T - 1$, the same argument implies that $p_j \in \{0, T - 1\}$. However jobs j with $\bar{p}_j = T$ and $T - 1 \leq p_j \leq T$ are not tested in Opt. Therefore increasing their processing time to $p_j = T$ does not change Opt but increases the cost of Random and therefore increases the competitive ratio.

In conclusion a worst case instance is described completely by the number of jobs n and fractions α, β, γ as follows, see Figure 4.

- A $1 - \alpha - \beta - \gamma$ fraction of the jobs have $\bar{p}_j = T$ and $p_j = 0$. (type 0 jobs)
- An α fraction of the jobs have $\bar{p}_j = T$ and $p_j = T$. (type T jobs)
- A β fraction of the jobs have $\bar{p}_j = E$ and $p_j = E$. (type E jobs)
- A γ fraction of the jobs have $\bar{p}_j = E + \epsilon$ and $p_j = E + \epsilon$ for some arbitrarily small $\epsilon > 0$. (type E+ jobs)

4.1.1 Cost of Random

Let n be the total number of jobs in the instance. In the following expressions for simplification we will omit ϵ. We denote by $L := n + T\alpha n + E\beta n$ the length of part (1). This means that for a job j of type $0, T$ or E, the expected time its test starts is $(L - 1 - p_j)/2$ and hence its expected completion time, which is $1 + p_j$ time units later,

is $(L+1+p_j)/2$. The expected objective value of RANDOM can be expressed as

$$\begin{aligned}\text{ALG} =&(1-\gamma)n(n+1+T\alpha n+E\beta n)/2 &(1)\\ &+T\alpha n/2+E\beta n/2 &(2)\\ &+\gamma n(n+T\alpha n+E\beta n) &(3)\\ &+E\gamma n(\gamma n+1)/2 &(4)\end{aligned}$$

where (1) is the sum of $(L+1)/2$ for all jobs completed in the first part, (2) is the additional part in the expected completion time for job types T and E. Jobs completed in the second part have all the same processing time. The i-th to be completed in part (2) has completion time $L+Ei$. Hence the total completion time of these jobs is expressed as the sum of the expressions (3) and (4).

4.1.2 Cost of OPT

By the *smallest processing time first rule*, the optimal schedule first tests and executes all type 0 jobs. Then it executes untested all type T, E and E^+ jobs in that order. Hence the optimal objective value is stated as follows, where every other expression represents the total completion times of some job type followed by the delay these jobs induce on subsequent job types.

$$\begin{aligned}\text{OPT} =&(1-\alpha-\beta-\gamma)n((1-\alpha-\beta-\gamma)n+1)/2+\\ &(1-\alpha-\beta-\gamma)n(\alpha+\beta+\gamma)n+\\ &T\alpha n(\alpha n+1)/2+\\ &T\alpha n(\beta+\gamma)n+\\ &E\beta n(\beta n+1)/2+\\ &E\beta n\gamma n+\\ &E\gamma n(\gamma n+1)/2.\end{aligned}$$

4.1.3 Competitive ratio

We say that fractions α, β, γ are *valid* iff $\alpha, \beta, \gamma \geq 0$ and $\alpha+\beta+\gamma \leq 1$. The algorithm is T-competitive if $T \cdot \text{OPT} - \text{ALG} \geq 0$ for all $n \geq 0$ and all valid fractions α, β, γ. The costs can be written as $\text{ALG} = \frac{n^2}{2}\text{ALG}_2 + \frac{n}{2}\text{ALG}_1$ and $\text{OPT} = \frac{n^2}{2}\text{OPT}_2 + \frac{n}{2}\text{OPT}_1$ for

$$\begin{aligned}\text{ALG}_2 =&1+\gamma+\beta E+\beta\gamma E+\gamma^2 E+\alpha T+\alpha\gamma T\\ \text{ALG}_1 =&1-\gamma+\beta E+\gamma E+\alpha T\\ \text{OPT}_2 =&1-\alpha^2-2\alpha\beta-\beta^2-2\alpha\gamma-2\beta\gamma-\gamma^2\\ &+\beta^2 E+2\beta\gamma E+\gamma^2 E+\alpha^2 T+2\alpha\beta T+2\alpha\gamma T\\ \text{OPT}_1 =&1-\alpha-\beta-\gamma+\beta E+\gamma E+\alpha T.\end{aligned}$$

It suffices to show separately the inequalities $T\cdot\text{OPT}_2-\text{ALG}_2 \geq 0$ and $T\cdot\text{OPT}_1-\text{ALG}_1 \geq 0$ for all valid α, β, γ fractions.

We start with the first inequality, and consider the following left hand side.

$$G = T[1+(\beta+\gamma)^2(E-1)+\alpha^2(T-1)+2\alpha(\beta+\gamma)(T-1)-\alpha-\alpha\gamma]-\gamma-1-E(\gamma^2+\beta\gamma+\beta).$$

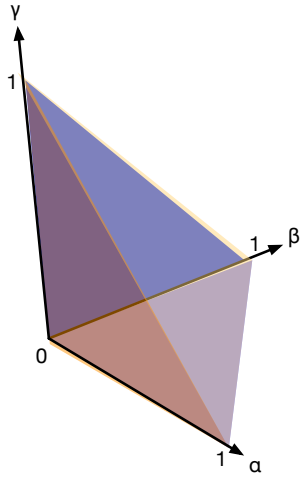

Fig. 5 Validity region for (α, β, γ).

4.1.4 Breaking into cases

We want to find parameters T, E with minimal T such that $G(T, E, \alpha, \beta, \gamma) \geq 0$ for all valid fractions, i.e. $\alpha, \beta, \gamma \geq 0$ with $\alpha + \beta + \gamma \leq 1$. We call this the *validity polytope* for α, β, γ, see Figure 5. For this purpose we made numerical experiments which gave us a range where the optima could belong, namely $T \in [1.71, 1.89], E \in [2.81, 2.89]$.

Our general approach consists in identifying values (α, β, γ) which are local minima for G. Each of these points (α, β, γ) generate conditions on T, E of the form $G(T, E, \alpha, \beta, \gamma) \geq 0$. The optimal pair (T, E) is then the pair with minimal T satisfying all the generated conditions.

The analysis follows a partition of the validity polytope. First we consider the open region $\{(\alpha, \beta, \gamma) | 0 < \alpha, 0 < \beta, 0 < \gamma, \alpha + \beta + \gamma < 1\}$. Then we consider the 4 open facets on the border defined by the equations $\alpha + \beta + \gamma = 1, \alpha = 0, \beta = 0, \gamma = 0$. Finally we consider the 6 closed edges that form the edges of the polytope. Note that the vertices of the polytope $(0,0,0), (0,0,1), (0,1,0), (1,0,0)$ belong each to several edges.

– Case 1: open polytope. The second order derivatives of G in α, β, γ are

$$\frac{\partial^2 G}{\partial^2 \alpha} = 2T(T-1)$$
$$\frac{\partial^2 G}{\partial^2 \beta} = 2T(E-1)$$
$$\frac{\partial^2 G}{\partial^2 \gamma} = 2T(E-1) - 2E$$

which are all positive in the considered T- and E-range. Hence a local minimum on the open polytope must be a point (α, β, γ) that is a root for the derivative in each of the 3 directions. Hence we choose α as the root

$$\alpha = \beta - \gamma + \frac{1+\gamma}{2(T-1)},$$

β as the root

$$\beta = \frac{1+\gamma-2\gamma T}{2T},$$

and γ as the root

$$\gamma = \frac{(E(T-1)-T)(2T-1)}{E(T-1)+T}.$$

For this point the condition $G \geq 0$ translates into the following condition on T, E.

$$E^2(T-1)^2 + T(2T-1) - ET^2 \geq 0. \tag{5}$$

- Case 2: facet $\alpha+\beta+\gamma = 1$. In that case the derivative of G in β is $1-\alpha(E-T)$. This means that G is linear in β, and a local minimum lies on the boundary of the triangle, which we considered open. Hence such a local minimum will be considered in a case below. Note that in the degenerate case $\alpha = 1/(E-T)$ the value of G is independent of β, hence it is enough to consider an equivalent point on the boundary.
- Case 3: facet $\gamma = 0$. In this case the extreme α value for G is

$$\alpha = \frac{1}{2(T-1)} - \beta,$$

and then the extreme β value for G is $\beta = 1/2T$. For this point the condition $G \geq 0$ translates into the following condition on T, E.

$$\frac{1}{T-1} + 4(T-1) - \frac{E}{T} \geq 0. \tag{6}$$

- Case 4: facet $\alpha = 0$. In this case the extreme β value for G is

$$\beta = \frac{E+\gamma E+2\gamma T-2\gamma ET}{2T(E-1)},$$

but then the second order derivative of G in γ is

$$\frac{\partial^2 G}{\partial^2 \gamma} = -\frac{E^2}{2T(E-1)}$$

which is negative. Hence local minimum of this triangle is on its boundary.
- Case 5: facet $\beta = 0$. The extreme α value for G is

$$\alpha = \frac{1+3\gamma-2\gamma T}{2(T-1)},$$

and then the extreme γ value for G is

$$\gamma = \frac{(2-T)(2T-1)}{4E(T-1^2)-T(5-4T(2-T))}.$$

But in the considered region for (T, E) the value of $\alpha+\gamma$ exceeds 1, and is therefore outside the boundaries of the triangle.
- Case 6: edge $(\alpha, \beta, \gamma) = (x, 1-x, 0)$ for $0 \leq x \leq 1$. The extreme point for x is

$$x = 1 - \frac{1}{2T}.$$

For this point the condition $G \geq 0$ translates into the following condition on T, E.

$$T(T-1) - \frac{3}{4} - \frac{E}{4T} \geq 0. \tag{7}$$

– Case 7: edge $(\alpha, \beta, \gamma) = (x, 0, 1-x)$ for $0 \leq x \leq 1$. The extreme point for x is

$$x = \frac{2ET + 2T - 2T^2 - 2E - 1}{2(E-T)(T-1)},$$

which generates the following condition

$$4E(1-(2-T)T^2) - (2T(T-1)-1)^2 \geq 0. \tag{8}$$

– Case 8: edge $(\alpha, \beta, \gamma) = (0, x, 1-x)$ for $0 \leq x \leq 1$. Here G is linear increasing in x, hence a local minimum is reached at $x = 0$, generating the condition

$$E(T-1) - 2 \geq 0. \tag{9}$$

– Case 9: edge $(\alpha, \beta, \gamma) = (x, 0, 0)$ for $0 \leq x \leq 1$. The extreme point for x is

$$x = \frac{1}{2(T-1)},$$

generating the condition

$$4T - 5 - \frac{1}{T-1} \geq 0. \tag{10}$$

– Case 10: edge $(\alpha, \beta, \gamma) = (0, x, 0)$ for $0 \leq x \leq 1$. The extreme point for x is

$$x = \frac{E}{2T(E-1)},$$

generating the condition

$$4(T-1) - \frac{E^2}{T(E-1)} \geq 0. \tag{11}$$

– Case 11: edge $(\alpha, \beta, \gamma) = (0, 0, x)$ for $0 \leq x \leq 1$. The extreme point for x is

$$x = \frac{1}{2(ET - E - T)},$$

generating the condition

$$T - 1 - \frac{1}{4(ET - E - T)} \geq 0. \tag{12}$$

In summary we want to find values T, E that satisfy all conditions (5) to (12) and minimize T. In the considered region for T and E, the conditions (7), (9), (10) and (11) are satisfied. Hence we focus on the remaining conditions, and find out that the optimal point lies on the intersection of the left hand sides of condition (6) and (8). The solutions are roots to a polynomial of degree 5, and in only one of them T is larger than the golden ratio, which it has to. Numerically we obtain the optimal parameters $T \approx 1.7453$ and $E \approx 2.8609$.

We conclude the proof by considering the inequality $T \cdot \textsc{Opt}_1 - \textsc{Alg}_1 \geq 0$ which is

$$\gamma(E-1)(T-1) + \beta(E-1)t + (1-\alpha(2-T))T - 1 - \beta E \geq 0.$$

Taking the derivative of the left hand side reveals that it is decreasing in α and increasing in β and γ for the chosen values T, E. Hence the expression is minimized at $\alpha = 1, \beta = 0, \gamma = 0$, where its value is $T(T-1) - 1 > 0$. Therefore we have shown the following theorem.

Theorem 10 *The competitive ratio of the algorithm* Random *is at most* 1.7453 *for scheduling with testing with the objective of minimizing the sum of completion times.*

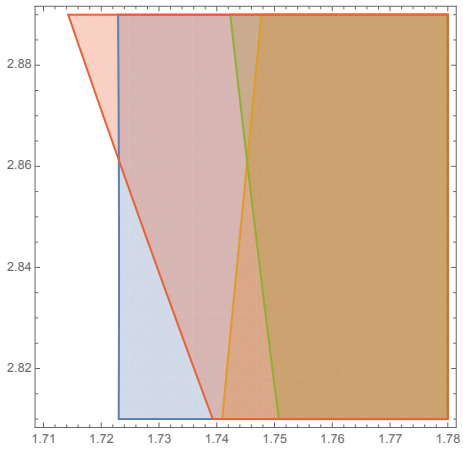

Fig. 6 Regions where conditions (5):blue, (6):orange, (8):green and (12):red are satisfied by points (T, E), with T ranging horizontally and E ranging vertically.

4.2 Lower bound for randomized algorithms

In this section we give a lower bound on the best possible competitive ratio of any randomized algorithm against an oblivious adversary. We do so by specifying a probability distribution over inputs and proving a lower bound on $E[\textsc{Alg}]/E[\textsc{Opt}]$ that holds for all deterministic algorithms Alg. By Yao's principle [8, 60] this gives the desired lower bound.

The probability distribution over inputs with n jobs has a constant parameter $0 < q < 1$ and is defined as follows: Each job j has upper limit $\bar{p}_j = 1/q > 1$, and its processing time p_j is set to 0 with probability q and to $1/q$ with probability $1 - q$.

Estimating $E[\textsc{Opt}]$. Let Z denote the number of jobs with processing time 0. Note that Z is a random variable with binomial distribution. The optimal schedule first tests and executes the Z jobs with $p_j = 0$ and then executes the $n - Z$ jobs with $p_j = 1/q$ untested. Hence, the objective value of Opt is:

$$\frac{Z(Z+1)}{2} + Z(n-Z) + \frac{(n-Z)(n-Z+1)}{2q}.$$

Using $E[Z] = nq$ and $E[Z^2] = (nq)^2 + nq(1-q)$, we obtain

$$E[\textsc{Opt}] = \frac{n^2}{2}\left(\frac{1}{q} + 3q - 2 - q^2\right) + O(n).$$

Estimating $E[\textsc{Alg}]$. First, observe that we only need to consider algorithms that schedule a job j immediately if the job has been tested and $p_j = 0$. Furthermore, we only need to consider algorithms that never create idle time before all jobs are completed.

We claim that any such algorithm satisfies $E[\textsc{Alg}] \geq \frac{n^2}{2q}$ for all n. We prove this by induction on n. Let $\textsc{Alg}(k)$ denote the objective value of the algorithm Alg executed for a random instance with k jobs that is generated by our probability distribution for

$n = k$ (i.e., all k jobs have $\bar{p}_j = 1/q$ and p_j is set to 0 with probability q and to $1/q$ otherwise).

Consider the base case $n = 1$. If ALG executes job 1 without testing, then $\text{ALG}(1) = 1/q$. If ALG tests the job and then necessarily executes it right away, since there are no other jobs, then $E[\text{ALG}(1)] = 1 + (q \cdot 0 + (1-q) \cdot (1/q)) = 1/q$. In both cases, $E[\text{ALG}(1)] = 1/q \geq \frac{n^2}{2q}$.

Now assume the claim has been shown for $n-1$, i.e., $E[\text{ALG}(n-1)] \geq \frac{(n-1)^2}{2q} = \frac{n^2}{2q} - n/q + \frac{1}{2q} > \frac{n^2}{2q} - n/q$. Consider the execution of ALG on an instance with n jobs, and make a case distinction on how the algorithm handles the first job it tests or executes. Without loss of generality, assume that this job is job 1.

- Case 1: ALG executes job 1 without testing (completing at time $C_1 = 1/q$), or it tests jobs 1 and then executes it immediately independent of its processing time (with expected completion time $E[C_1] = 1+(1-q)/q = 1/q$). After the completion of job 1, the algorithm schedules the remaining $n-1$ jobs, which is a random instance with $n-1$ jobs. Hence, the objective value is $E[C_1] + E[C_1](n-1) + E[\text{ALG}(n-1)] = 1/q + (n-1)/q + E[\text{ALG}(n-1)] \geq n/q + \frac{n^2}{2q} - n/q = \frac{n^2}{2q}$.
- Case 2: ALG tests job 1 and then executes it immediately if its processing time is 0, but defers it if its processing time is $1/q$. Assume first that if $p_1 = 1/q$, then ALG defers the execution of p_1 to the very end of the schedule. We have

$$E[\text{ALG}(n)|p_1 = 0] = 1 + (n-1) + E[\text{ALG}(n-1)]$$

and

$$E[\text{ALG}(n)|p_1 = 1/q] = n + E[\text{ALG}(n-1)] + E[\mathit{len}(\text{ALG}(n-1))] + 1/q,$$

where $\mathit{len}(\text{ALG}(n-1))$ is the length of the schedule for $n-1$ jobs. Note that every job contributes $1/q$ to the expected schedule length no matter whether it is tested (in which case it requires time 1 for testing and an additional expected $(1-q)/q$ time for processing) or not (in which case its processing time is $1/q$ for sure). Therefore, $E[\mathit{len}(\text{ALG}(n-1))] = (n-1)/q$. So we have:

$$\begin{aligned} E[\text{ALG}(n)] &= q(n + E[\text{ALG}(n-1)]) + (1-q)(n + n/q + E[\text{ALG}(n-1)]) \\ &= qn + n + n/q - qn - n + E[\text{ALG}(n-1)] \\ &= n/q + E[\text{ALG}(n-1)] \\ &\geq \frac{n^2}{2q}. \end{aligned}$$

Finally, we need to consider the possibility that $p_1 = 1/q$ and ALG defers job 1, but schedules it at some point during the schedule for the remaining $n-1$ jobs instead of at the very end of the schedule. Assume that ALG schedules job 1 in such a way that k of the remaining $n-1$ jobs are executed after job 1. We compare this schedule to the schedule where job 1 is executed at the very end of the schedule. Let K be the set of k jobs that are executed after job 1 by ALG. Note that the jobs in the set K can be jobs that are scheduled without testing (and thus executed with processing time $1/q$), jobs that are tested and executed after the execution of job 1 (so that the expected time for testing and executing them is $1/q$), or jobs that are tested before the execution of job 1 but executed afterwards (in which case

their processing time must be $1/q$, since jobs with processing time 0 are executed immediately after they are tested). Hence, moving the execution of job 1 from the very end of the schedule ahead of k job executions will change the expected objective value as follows: The expected completion time of job 1 decreases by k/q, and the completion time of each of the k jobs in K increases by $1/q$. Therefore, $E[\textsc{Alg}(n)]$ is the same as when job 1 is executed at the end of the schedule, and we get $E[\textsc{Alg}(n)] \geq \frac{n^2}{2q}$ as before.

Theorem 11 *No randomized algorithm can achieve a competitive ratio less than* 1.6257 *for scheduling with testing with the objective of minimizing the sum of completion times.*

Proof Since we have $E[\textsc{Opt}] = \frac{n^2}{2}\left(\frac{1}{q} + 3q - 2 - q^2\right) + O(n)$ and $E[\textsc{Alg}] \geq \frac{n^2}{2q}$, Yao's principle [8, 60] gives a lower bound that is arbitrarily close (for large enough n) to

$$\frac{1/q}{1/q + 3q - 2 - q^2}$$

for randomized algorithms against an oblivious adversary. The bound is maximized for $q = 1 - 1/\sqrt{3} \approx 0.42265$, giving a lower bound of 1.62575. □

5 Deterministic Algorithms for Uniform Upper Limits

In this section we investigate the problem of scheduling with testing on instances in which all jobs have a uniform upper limit $\bar{p}$. In Subsection 5.1, we give a deterministic algorithm that achieves a ratio strictly less than 2. In Subsection 5.2, we study an even further restricted class of *extreme uniform* instances that consist of jobs with uniform upper limit $\bar{p}$ and processing times in $\{0, \bar{p}\}$. We give an algorithm with improved competitive ratio that is particularly interesting as it is near-optimal algorithm for the class of worst-case instances for deterministic algorithm from Theorem 9 in Subsection 3.2.

5.1 An improved algorithm for uniform upper limits

We assume that all jobs have upper limit $\bar{p}$. We design an algorithm with a competitive ratio strictly less than 2 by combining Threshold, presented in Section 3.1, with a new algorithm Beat. The new algorithm Beat performs well on instances with upper limit roughly 2, but its performance becomes worse for larger upper limits. Therefore, we employ in this case the algorithm Threshold.

To simplify the analysis, we consider the limit of $\textsc{Alg}(I)/\textsc{Opt}(I)$ when the number n of jobs approaches infinity. We say that an algorithm Alg is *asymptotically ρ_∞-competitive* or *has asymptotic competitive ratio at most ρ_∞* if

$$\lim_{n\to\infty} \sup_I \textsc{Alg}(I)/\textsc{Opt}(I) \leq \rho_\infty.$$

Algorithm 3 (Beat) *The algorithm* Beat *balances the time testing jobs and the time executing jobs while there are untested jobs. A job is called* short *if its running time is at most $E = \max\{1, \bar{p}-1\}$, and* long *otherwise. Let* TotalTest *denote the time we spend testing long jobs and let* TotalExec *be the time long jobs are executed. We iterate testing an arbitrary job and then execute the job with smallest processing time*

either, if it is a short job, or if TotalExec $+ p_k$ *is at most* TotalTest. *Once all jobs have been tested, we execute the remaining jobs in order of non-decreasing processing time. The pseudocode is shown in Pseudocode 1.*

Pseudocode 1: BEAT

```
Output: A schedule of tests and executions of all jobs.
TotalTest ← 0; //total time of executed tests of long jobs
TotalExec ← 0; //total time of executed long jobs
while there are untested jobs do
    k ← tested, not executed job with minimum p_k; //p_k = ∞ if no such job
    if TotalExec + p_k ≤ TotalTest then
        execute k;
        TotalExec ← TotalExec + p_k;
    else
        j ← an arbitrary untested job ;
        test j;
        if p_j ≤ E then
            execute j;
        else
            TotalTest ← TotalTest + 1;
execute all remaining jobs in order of non-decreasing p_j;
```

We will analyze algorithm BEAT in Section 5.1.1. In Lemma 13 we will make a structural observation about the algorithm schedule for a worst-case instance. In Lemma 15 we prove that the asymptotic competitive ratio of BEAT for $\bar{p} < 3$ is at most

$$\rho_{\infty}^{BEAT} = \frac{1 + 2(-2+\bar{p})\bar{p} + \sqrt{(1-2\bar{p})^2(-3+4\bar{p})}}{2(-1+\bar{p})\bar{p}}.$$

This function decreases, when $\bar{p}$ increases. Alternatively, for small upper limit we can execute each job without test. Then there is a worst-case instance where all jobs have processing time $p_j = 0$. The optimal schedule tests each job only if the upper limit $\bar{p}$ is larger than one and executes it immediately. For $\bar{p} < 1$ this means the competitive ratio is 1 and otherwise it is $\bar{p}$, which monotonically increases. Thus, we choose a threshold $T_1 \approx 1.9338$ for $\bar{p}$, where we start applying BEAT: the fixpoint of the function ρ_{∞}^{BEAT}.

For upper limits $\bar{p} > 3$, the performance behavior of BEAT changes and the asymptotic competitive ratio increases. Thus, we employ the algorithm THRESHOLD for large upper limits. Recall from Section 3.1 that for $\bar{p} > 2$ THRESHOLD tests all jobs, executes those with $p_j \leq 2$ immediately and defers the other jobs. In Subsection 5.1.2, we argue that there is a worst-case instance with short jobs that have processing time 0 or 2 and long jobs with processing time $\bar{p}_j = \bar{p}$ and that no long job is tested in an optimal solution. This allows us to prove in Theorem 17 that the competitive ratio for THRESHOLD is at most

$$\rho_{\infty}^{THRESH} = \begin{cases} \frac{-3+\bar{p}+\sqrt{-15+\bar{p}(18+\bar{p})}}{2(\bar{p}-1)} & \text{if } \bar{p} \in (2,3) \\ \sqrt{3} \approx 1.73 & \text{if } \bar{p} \geq 3. \end{cases}$$

The function for small $\bar{p}$ is a monotone function decreasing from 2 to $\sqrt{3}$ in the limits for $\bar{p} \in (2,3)$. We choose a threshold, where we change from applying BEAT to employing

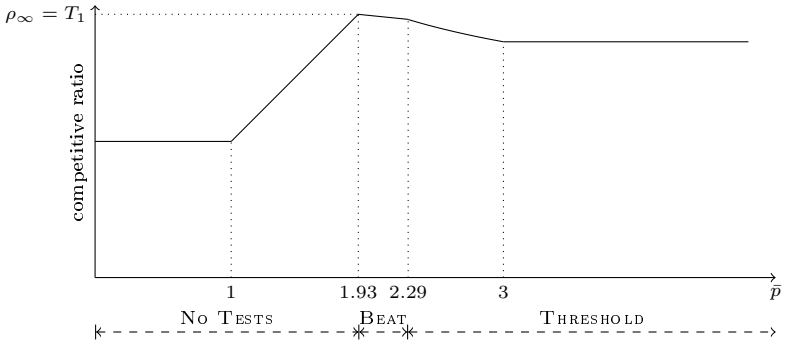

Fig. 7 Competitive ratio depending on $\bar{p}$.

Threshold at $T_2 \approx 2.2948$, the crossing point of the two functions describing the competitive ratio of Beat and Threshold in $(2, 3)$.

Algorithm 4 *Execute all jobs without testing them, if the upper limit $\bar{p}$ is less than $T_1 \approx 1.9338$. Otherwise, if the upper limit $\bar{p}$ is greater than $T_2 \approx 2.2948$, execute the algorithm* Threshold. *For upper limits between T_1 and T_2, execute the Algorithm* Beat.

The function describing the asymptotic competitive ratio depending on $\bar{p}$ is displayed in Figure 7. Its maximum is attained at T_1, which is a fixpoint. Thus we obtain the following result.

Theorem 12 *For scheduling with testing with the objective of minimizing the sum of completion times, the asymptotic competitive ratio of Algorithm 4 on instances with uniform upper limits is $\rho_\infty = T_1 \approx 1.9338$, which is the only real root of $2\bar{p}^3 - 4\bar{p}^2 + 4\bar{p} - 1 - \sqrt{(1-2\bar{p})^2(4\bar{p}-3)}$.*

5.1.1 Analysis of Beat

We first make a structural observation about the algorithm schedule for a worst-case instance.

Lemma 13 *There are worst-case uniform instances for* Beat *in which the jobs are tested in order of decreasing p_j, at most one job has $p_j \in (E, \bar{p})$, and all other jobs have $p_j \in \{0, E, \bar{p}\}$.*

Proof Let an arbitrary worst-case instance be given. Recall that a job is called *short* if its running time is at most $E = \max\{1, \bar{p} - 1\}$, and *long* otherwise. We first argue that the short jobs are tested last. If not, the test and execution of some short job j_s is followed by the test of a long job j_l. If j_l is not executed immediately after its test, moving the test of j_l in front of the test of j_s increases the cost of the algorithm by 1. If j_l is executed immediately after its test, moving the test and execution of j_l in front of the test of j_s increases the cost of the algorithm by $p_{j_l} - p_{j_s} > 0$. Hence, in a worst-case instance the short jobs are tested after all the tests of long jobs.

We call a long job an *executed long job* if it is executed by Beat in line 6 of Pseudocode 1, and a *delayed long job* or *delayed job* if it is excuted in line 15. The long

jobs have processing time larger than $E \geq \bar{p} - 1$, which means they are not tested by Opt. Hence, increasing the processing time of a long job does not increase the optimal cost. For the delayed jobs, increasing their processing time to $\bar{p}$ increases the algorithm cost, but does not change the schedule, so in a worst-case instance we can assume that all delayed jobs have $p_j = \bar{p}$.

For the executed long jobs, note that no two jobs are executed without a test in between, as their processing time is larger than one, the length of a test. We claim that we can assume that each executed long job is tested immediately before its execution. If not, consider an executed long job j that was tested earlier and is executed immediately after the test of another long job j'. Note that $p_{j'} \geq p_j$ and that all long jobs j'' executed between the test of j and the execution of j satisfy $p_{j''} \leq p_j$. Hence, we can swap the tests of j and j' without affecting the schedule.

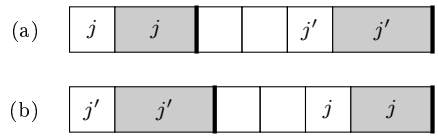

Fig. 8 (a) Long job j with $p_j < p_{j'}$ is executed before j'; (b) the tests (and executions) of j and j' have been swapped.

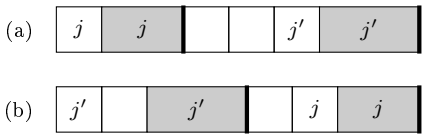

Fig. 9 (a) Long job j with $p_j < p_{j'}$ is executed before j'; (b) the tests and executions of j and j' have been swapped, and the execution of j' has moved after the test of a long delayed job.

Next, we claim that we can assume that the executed long jobs are tested in order of decreasing processing times. If not, there must be an executed long job j that precedes an executed long job j' (potentially with some tests of delayed long jobs in between) such that $p_j < p_{j'}$. Swap the tests of j and j'. If job j' is still executed immediately after its test in the new position (see Figure 8), the cost of the algorithm increases by $p_{j'} - p_j$. If job j' is executed only after a further test of a delayed job (see Figure 9; this happens if TotalExec $+ p_{j'} >$ TotalTest holds after testing j'), the cost of the algorithm increases by $1 + (p_{j'} - p_j)$. Note that the execution of j' cannot move behind two or more tests of delayed jobs because $E < p_j < p_{j'} \leq \bar{p}$ implies $p_{j'} - p_j < 1$. As the cost of the algorithm increases in both cases while the optimal cost remains unchanged, the executed long jobs must indeed be tested in order of decreasing processing times in a worst-case instance.

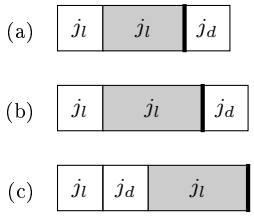

Fig. 10 (a) The execution of the last long executed job j_l is followed by the test of a delayed job j_d; (b) increasing p_{j_l} increases the cost of the algorithm; (c) if the execution of j_l moves after the test of j_d, the increase in cost is even larger.

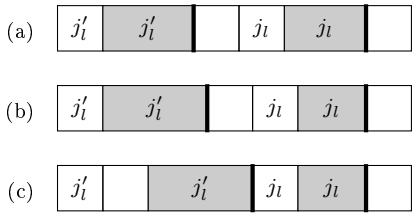

Fig. 11 (a) The execution of the last long executed job j_l is followed by the test of a short job; (b) increasing $p_{j'_l}$ and decreasing p_{j_l} increases the cost of the algorithm; (c) if the execution of j'_l moves after the test of a delayed long job, the increase in cost is even larger.

Now, we want to show that we can also assume that the processing times of all the executed long jobs (with at most one exception) are equal to $\bar{p}$. Consider the last executed long job j_l, and assume that $p_{j_l} < \bar{p}$ (otherwise, all executed long jobs have processing time $\bar{p}$). Case 1: If j_l is followed by the test of a delayed long job j_d, we increase p_{j_l} to $\bar{p}$, an increase of less than 1. After this increase, j_l will either still be executed immediately after its test, or it will be executed after the test of j_d (see Figure 10). The cost of the algorithm has thus increased by at least $\bar{p} - p_{j_l} > 0$. Case 2: If the execution of the last executed long job is followed by the test of a short job and there is at least one other executed long job with processing time strictly less than $\bar{p}$, we proceed as follows: We shift processing time from the last executed long job j_l to the one before, say j'_l, until either $p_{j'_l} = \bar{p}$ or $p_{j_l} = E$. This increases the completion time of the first of the two jobs (the execution of that job may potentially also move after the test of a delayed long job), but does not change the completion time of any other job (see Figure 11). If p_{j_l} becomes equal to E, the job j_l becomes a short job, but the schedule of the algorithm does not change. Thus, in both cases the cost of the algorithm can be increased while keeping the optimal cost unchanged, a contradiction to the instance being a worst-case instance. Hence, neither Case 1 nor Case 2 can apply in a worst-case instance, and therefore we have at most one executed long job with processing time strictly less than $\bar{p}$, and that job (if it exists) is tested last among all long jobs.

Finally we observe that both the algorithm and the optimal schedule test all short jobs with $p_j \in [0, \bar{p}-1]$ independent of their actual processing time. Also the execution order of the algorithm and the optimal schedule solely depend on the ordering of the processing times. Therefore Lemma 3 implies that we can assume that the short jobs have processing times either 0 or $\bar{p} - 1$. Next, observe that increasing the processing times of all short jobs with processing times in $[\bar{p} - 1, E]$ to E does not change the optimal cost as Opt can execute these jobs untested (recall that a job with $p_j = \bar{p} - 1$ takes time $\bar{p}$ no matter whether it is tested and executed, or executed untested). It increases the algorithm cost, however. Thus, we can assume that in a worst-case instance all short jobs have $p_j \in \{0, E\}$. It is also clear that in a worst-case instance the short jobs are tested in order of decreasing processing times by the algorithm, and hence all jobs are tested in order of decreasing processing times (first the long jobs with processing time $\bar{p}$, then possibly the one long job with processing time between E and $\bar{p}$, and finally the short jobs). □

Consequently, the schedule produced by Beat consists of the following parts (in this order), see also Figure 12:

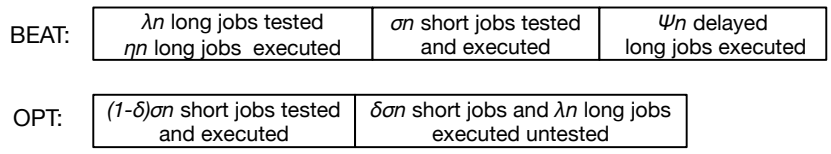

Fig. 12 Structure of schedules produced by Beat and Opt.

- The tests of the λ fraction of jobs, that are long jobs, interleaved with executions of the η fraction of all jobs, that are also long jobs and that are executed during the "while there are untested jobs" loop.
- The tests and immediate executions of the short jobs, which is a $\sigma = 1 - \lambda$ fraction of all jobs. Let δ be the fraction of short jobs with $p_j = E$.
- The executions of the $\psi = \lambda - \eta$ fraction of jobs, that are delayed long jobs, in the "execute all remaining jobs" statement.

Opt consists of the following parts (in this order), see also Figure 12:

- The tests and immediate executions of the $(1-\delta)\sigma$ fraction of jobs that are short and have processing time 0.
- The untested executions of the $\delta\sigma$ fraction of jobs which are short and have $p_j = E$ and the λ fraction of jobs that are long.

We note that TotalTest has value λn when all long jobs are tested, so the total execution time in Phase 1, which is at least $\bar{p}(n\eta - 1) + E$ by Lemma 13, cannot exceed λn. As long jobs have $p_j > E \geq 1$, there are always at least as many long jobs tested as are executed. Thus, TotalExec never decreases below TotalTest $- \bar{p}$, as then some job can be executed. Hence, we have

$$\bar{p}\eta \leq \lambda + O(1/n) < \bar{p}\eta + O(1/n). \tag{13}$$

Furthermore, we have $\lambda = \eta + \psi$, which yields

$$\psi \leq (1 - 1/\bar{p})\,\lambda + O(1/n). \tag{14}$$

We first consider the algorithm schedule.

Lemma 14 *For a fraction $\delta \in [0,1]$ of short jobs with processing time $p_j = E$, we can bound the algorithm cost by*

$$\begin{aligned}\textsc{Alg} \leq \frac{n^2}{2}\Bigg[&\lambda^2\left(\bar{p} + 2 - \frac{1}{\bar{p}}\right) + \sigma^2((1+E)(2\delta - \delta^2) + (1-\delta)^2) \\ &+ 2\lambda\sigma\left(2 + \left(1 - \frac{1}{\bar{p}}\right)(1 + E\delta)\right)\Bigg] + O(n).\end{aligned}$$

Proof There is an η fraction of jobs completed in the first part, each executed when TotalExec$+p_j \leq$ TotalTest in the algorithm. Thus, the completion time of the i-th such job is at most $2i\bar{p} + 1$. The sum of these completion times is $\bar{p}\eta^2 n^2 + O(n)$. A fraction of $\delta\sigma$ jobs is short and has $p_j = E$. They are executed before the other $(1-\delta)\sigma$

fraction of jobs with $p_j = 0$ is executed. This means the completion times of the short jobs contribute

$$\frac{n^2}{2}\left[(1+E)\delta^2\sigma^2 + (1-\delta)^2\sigma^2 + 2(1+E)\delta\sigma(1-\delta)\sigma\right] + O(n)$$
$$= \frac{n^2}{2}\left[\sigma^2((1+E)(2\delta-\delta^2) + (1-\delta)^2)\right] + O(n).$$

Additionally there is an ψ fraction of jobs, which are executed at the end of the schedule, each with processing time $\bar{p}$. Thus their contribution to the algorithm cost is $\bar{p}\psi^2 n^2/2 + O(n)$. The execution of the fraction σ of short jobs starts latest at time $n\lambda + \bar{p}n\eta$, and the execution of the fraction ψ of jobs is delayed by at most $n\lambda + \bar{p}n\eta + (1+E\delta)n\sigma$. Thus, the total objective value of Beat is at most:

$$\textsc{Alg} \le \frac{n^2}{2}\left[2\bar{p}\eta^2 + \sigma^2((1+E)(2\delta-\delta^2) + (1-\delta)^2) + \bar{p}\psi^2\right.$$
$$\left. +2(\lambda+\bar{p}\eta)\sigma + 2(\lambda + \bar{p}\eta + (1+E\delta)\sigma)\psi\right] + O(n).$$

By (13) and (14), we know that $\eta \le \lambda/\bar{p} + O(1/n)$ and $\psi \le (1-1/\bar{p})\,\lambda + O(1/n)$. Together with $\eta + \psi = \lambda$, this yields the desired bound. □

Lemma 15 *For uniform upper limit $\bar{p} \in [1.5,\ 3]$, the asymptotic competitive ratio of* Beat *is at most*

$$\frac{1 + 2(-2+\bar{p})\bar{p} + \sqrt{(1-2\bar{p})^2(-3+4\bar{p})}}{2(-1+\bar{p})\bar{p}}.$$

Proof We bounded the algorithm cost in Lemma 14 and thus first consider the optimal cost. In Opt, first a fraction $(1-\delta)\sigma$ of the short jobs is tested and executed with processing time 0. Then the remaining fraction $\delta\sigma$ of short jobs is executed with processing time $\bar{p}$ without test. Thus their contribution to the sum of completion times is

$$\frac{n^2}{2}\left[\sigma^2\left((1-\delta)^2 + \bar{p}\delta^2 + 2\delta(1-\delta)\right)\right] + O(n) \;= \frac{n}{2}\left[\sigma^2\left((\bar{p}-1)\delta^2 + 1\right)\right] + O(n).$$

All long jobs are executed untested at the end of the schedule and take $\bar{p}$ time units. Their sum of completion times is $\bar{p}\lambda^2 n^2/2 + O(n)$ and they are each delayed by $\sigma n(1 + (\bar{p}-1)\delta))$, giving:

$$\textsc{Opt} = \frac{n^2}{2}\left[\lambda^2\bar{p} + \sigma^2((\bar{p}-1)\delta^2 + 1) + 2\lambda\sigma(1 + (\bar{p}-1)\delta)\right] + O(n).$$

Then the asymptotic competitive ratio ρ_∞ for upper limit $\bar{p}$ in $[1.5, 3]$

$$\rho_\infty = \frac{\lambda^2\left(\bar{p} + 2 - \frac{1}{\bar{p}}\right) + \sigma^2((1+E)(2\delta-\delta^2) + (1-\delta)^2) + 2\lambda\sigma(2 + \left(1 - \frac{1}{\bar{p}}\right)(1+E\delta))}{\bar{p}\lambda^2 + \sigma^2((\bar{p}-1)\delta^2 + 1) + 2\lambda\sigma(1 + (\bar{p}-1)\delta)}.$$

For $\sigma = 0$ or $\lambda = 0$ this fulfills the claim. For the other values we set $\sigma = \alpha\lambda$ so the ratio becomes:

$$\frac{\bar{p} + 2 - \frac{1}{\bar{p}} + \alpha^2((1+E)(2\delta-\delta^2) + (1-\delta)^2) + 2\alpha(2 + \left(1 - \frac{1}{\bar{p}}\right)(1+E\delta))}{\bar{p} + \alpha^2((\bar{p}-1)\delta^2 + 1) + 2\alpha(1 + (\bar{p}-1)\delta)}.$$

We take the term to Mathematica to find the best bounds for it. For the case $1.5 < \bar{p} < 2$ we show that the adversary chooses $\delta = 0$ and α such that the first derivative in α equals 0. Otherwise, in the case $2 \leq \bar{p} \leq 3$, we show for $\delta = 0$ that we get exactly the same expression as for $\bar{p} < 2$. We prove the adversary chooses this case, which means the competitive ratio is bounded by the following function

$$\frac{1 + 2(-2 + \bar{p})\bar{p} + \sqrt{(1 - 2\bar{p})^2(-3 + 4\bar{p})}}{2(-1 + \bar{p})\bar{p}}.$$

□

5.1.2 Analysis of Threshold *for uniform* $\bar{p}$

In this section we analyze Algorithm Threshold (see Section 3.1) for instances with uniform upper limit $\bar{p} > 2$ and derive a competitive ratio as a function of $\bar{p}$.

Recall that for $\bar{p} > 2$, Threshold tests all jobs. It executes a job immediately if $p_j \leq 2$, and defers it otherwise. We have proved in Lemma 4 that we may assume that all jobs with $p_j \leq 2$ have execution times either 0 or 2. We also argued that in a worst case, Threshold tests first all long jobs, i.e., jobs j with $p_j > 2$, then follow the short jobs with tests (first length-2 jobs and then length-0 jobs), and finally Threshold executes the deferred long jobs in increasing order of processing times.

An optimum solution tests a job j only if $p_j + 1 < \bar{p}$. We show next that such long jobs to be tested in an optimal solution do not exist.

Lemma 16 *There is a worst-case uniform instance with short jobs that have processing times* 0 *or* 2 *and long jobs with processing time* $p_j = \bar{p}$*. Furthermore, none of the long jobs is tested in an optimal solution.*

Proof Consider an instance with short jobs that have processing times 0 or 2 (Lemma 4). We may increase the processing time of untested long jobs to their upper limit $\bar{p}$ without changing the optimal schedule. This cannot decrease the worst-case ratio as the algorithm's objective value can only increase.

It remains to consider the long jobs that are tested by an optimal solution. We show that we may assume that those do not exist. This is trivially true if $2 < \bar{p} < 3$. Then testing a long job j costs $1 + p_j > 3$ which is greater than running the job untested at $\bar{p} < 3$, and thus, an optimal solution would never test it.

Assume now that $\bar{p} \geq 3$. Threshold schedules any long job *after* all short jobs; first it runs long tested jobs with total execution time $1 + p_j < \bar{p}$ in non-decreasing order of p_j and then the untested jobs with execution time $\bar{p}$. As all untested jobs have processing time $p_j = \bar{p}$, we may assume that the algorithm and the optimum schedule long jobs in the same order. Reducing the processing times of all tested long jobs to $2 + \varepsilon$ for infinitesimally small $\varepsilon > 0$ does not change the schedule for any of the two algorithms, and thus, by Proposition 2, the ratio of the objective values of the algorithm and the optimum does not decrease.

Now, we argue that reducing the processing times of tested long jobs from $2+\varepsilon$ to 2 (thus making them short jobs) does not affect the optimal objective value, because ε is infinitesimally small, and can only increase the objective value of the algorithm. Consider the first long job that is tested by the optimum and the algorithm, say job ℓ. Consider the worst-case schedule of our algorithm for the new instance in which ℓ is turned into a short job with effectively the same processing time. The job ℓ is tested

and scheduled just before the short jobs with $p_j = 0$ instead of after them. Let a be the number of those short jobs. Then this change in p_ℓ to 2 improves the completion time of job ℓ by a and increases the completion time of a jobs by 2, so the net change in the objective value of the algorithm is $2a - a = a \geq 0$. The argument can be repeated until no tested long jobs are left. □

Theorem 17 *For scheduling with testing jobs with uniform upper limit $\bar{p} > 2$ with the objective of minimizing the sum of completion times, Algorithm* Threshold *has an asymptotic competitive ratio at most*

$$\rho_\infty = \begin{cases} \frac{-3+\bar{p}+\sqrt{-15+\bar{p}(18+\bar{p})}}{2(\bar{p}-1)} & \text{if } \bar{p} \in (2,3) \\ \sqrt{3} \approx 1.73 & \text{if } \bar{p} \geq 3. \end{cases}$$

The function for small $\bar{p}$ is a monotone function decreasing from 2 to $\sqrt{3}$ in the limits for $\bar{p} \in (2,3)$.

Proof Consider a worst-case instance according to Lemma 16. Let αn denote the number of short jobs of length 0, let βn be the number of short jobs of length 2, and let γn be the number of long jobs with $p_j = \bar{p}$. There are no other jobs, so $\alpha + \beta + \gamma = 1$. Recall, that we may assume that Threshold's schedule is as follows: first γn tests, βn tests and executions of length-2 jobs, then tests and executions of αn length-0 jobs, followed by the execution of long jobs with $p_j = \bar{p}$. The objective value is

$$\textsc{Alg} = n^2 \left(\gamma(\alpha+\beta+\gamma) + \frac{\beta^2}{2} \cdot 3 + 3\beta(\alpha+\gamma) + \frac{\alpha^2}{2} + \alpha\gamma + \frac{\gamma^2}{2} \cdot \bar{p} \right) + O(n). \quad (15)$$

To estimate the objective value of an optimal solution, we distinguish two cases for the upper limit $\bar{p}$.

Case: $\bar{p} > 3$. In this case, an optimal solution would test all short jobs, first the length-0 jobs and then the length-2 jobs. Then follow all long jobs without testing them (Lemma 16). Using the above notation, we have an optimal objective value

$$\textsc{Opt} = n^2 \left(\frac{\alpha^2}{2} + \alpha(\beta+\gamma) + \frac{\beta^2}{2} \cdot 3 + 3\beta\gamma + \frac{\gamma^2}{2} \cdot \bar{p} \right) + O(n).$$

Using $\gamma = 1-\alpha-\beta$, the asymptotic competitive ratio for any $\bar{p} > 3$ can be bounded by

$$\frac{2 - \alpha^2 - 2\alpha\beta + (4-3\beta)\beta + \bar{p}(-1+\alpha+\beta)^2}{-\alpha^2 + \alpha(2-6\beta) - 3(-2+\beta)\beta + \bar{p}(-1+\alpha+\beta)^2},$$

which has its maximum at $\sqrt{3}$ for $\alpha = (3-\sqrt{3})/2$ and $\beta = (\sqrt{3}-1)/2$.

Case: $\bar{p} \leq 3$. In this case, an optimal solution tests only short jobs with $p_j = 0$ and executes all other jobs untested, also short jobs with $p_j = 2$. The value of an optimum schedule is

$$\textsc{Opt} = n^2 \left(\alpha^2/2 + \alpha(\beta+\gamma) + \bar{p} \cdot \beta^2/2 + \bar{p} \cdot \beta\gamma + \bar{p} \cdot \gamma^2/2 \right) + O(n).$$

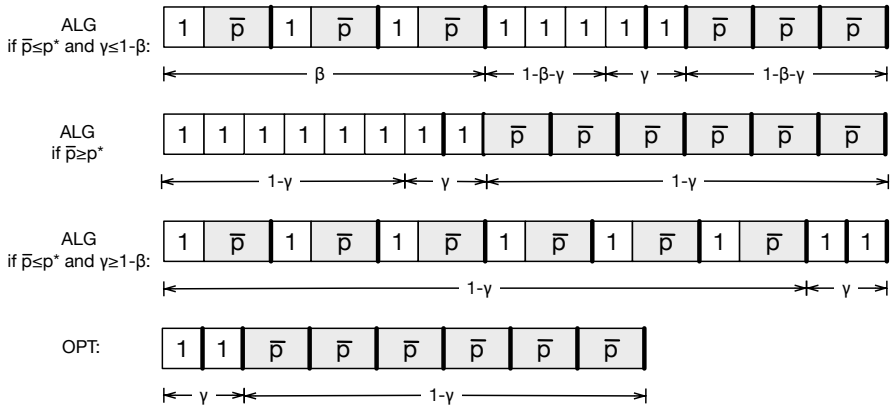

Fig. 13 The schedule produced by UTE and the optimal schedule.

With the value of THRESHOLD's solution given by Equation (15), the asymptotic competitive ratio is

$$\rho_\infty = \frac{\alpha^2 + 3\beta^2 + 8\beta\gamma + \alpha(6\beta + 4\gamma) + \gamma^2(2 + \bar{p})}{\alpha^2 + 2\alpha(\beta + \gamma) + (\beta + \gamma)^2\bar{p}}.$$

Using Mathematica we verify that this ratio has its maximum at the desired value

$$\frac{-3 + \bar{p} + \sqrt{-15 + \bar{p}(18 + \bar{p})}}{2(\bar{p} - 1)}.$$

□

5.2 A nearly optimal deterministic algorithm for extreme uniform instances

We present a deterministic algorithm for the restricted class of *extreme uniform* instances, that is almost tight for the instance that yields the deterministic lower bound. An *extreme uniform* instance consists of jobs with uniform upper limit $\bar{p}$ and processing times in $\{0, \bar{p}\}$. Our algorithm UTE requires a parameter $\rho \geq 1$ and attains competitive ratio $\rho \approx 1.8668$ for this class of instances when setting the algorithm parameter ρ accordingly.

Algorithm 5 (UTE) *Let parameter $\rho \geq 1$ be given. If the upper limit $\bar{p}$ is at most ρ, then all jobs are executed without test. Otherwise, all jobs are tested. The first* $\max\{0, \beta\}$ *fraction of the jobs are executed immediately after their test. The remaining fraction of the jobs are executed immediately after their test if they have processing time* 0 *and are delayed otherwise, see Figure 13. The parameter β is defined as*

$$\beta = \frac{1 - \bar{p} + \bar{p}^2 - \rho + 2\bar{p}\rho - \bar{p}^2\rho}{1 - \bar{p} + \bar{p}^2 - \rho + \bar{p}\rho}. \tag{16}$$

The choice of β will become clear in the analysis of the algorithm.

Theorem 18 *For scheduling with testing to minimize the sum of completion times, the competitive ratio of* UTE *on extreme uniform instances is at most* $\rho = \frac{1+\sqrt{3+2\sqrt{5}}}{2} \approx 1.8668$.

Proof If the upper limit $\bar{p}$ is at most ρ, by Lemma 1 the algorithm has competitive ratio $\bar{p}$, which fulfills the claim. Thus, we assume in the following $\bar{p} \geq \rho$. An instance is defined by the job number n, an upper limit $\bar{p}$ and a fraction γ such that the first $1-\gamma$ fraction of the jobs tested by UTE have processing time $\bar{p}$, while the jobs in the remaining γ fraction have processing time 0. The algorithm chooses β so as to have the smallest ratio ρ.

With the chosen fixed value of ρ, the value β from equation (16) is a decreasing function in $\bar{p}$ for $\bar{p} \geq \rho$. Hence there is a threshold value p^* such that $\beta(\bar{p}) \leq 0$ for all $\bar{p} \geq p^*$, which is

$$p^* := \frac{2\rho + \sqrt{4\rho - 3} - 1}{2(\rho - 1)} \approx 2.7961.$$

As in previous proofs, we start to analyze the ratio only for the n^2 dependent part of the costs of UTE and OPT. We distinguish three cases, depending on the ranges of $\bar{p}$ and γ.

Case $\rho \leq \bar{p} \leq p^*$ *and* $\gamma \leq 1 - \beta$. Consider for now β and ρ as some undetermined parameters which will be optimized in the analysis of this case. The optimal cost is

$$\textsc{Opt} = \gamma^2/2 + \bar{p}(\gamma - 1)^2/2 + \gamma(1 - \gamma)$$

while the cost of UTE is

$$\begin{aligned}\textsc{Alg} =& (\bar{p}+1)\beta^2/2 + \\ & \bar{p}(1-\beta-\gamma)^2/2 + \\ & \gamma^2/2 + \\ & (1-\gamma+\bar{p}\beta)\gamma + \\ & (1+\bar{p}\beta)(1-\beta-\gamma).\end{aligned}$$

The algorithm is ρ-competitive in this case if $g \geq 0$ for

$$g := 2(\rho\textsc{Opt} - \textsc{Alg}) = -2 - \beta^2 + \beta(2 - 2\gamma\bar{p}) + \gamma^2(\bar{p}-1)(\rho-1) + \bar{p}(\rho-1) + 2\gamma(\bar{p} + \rho - \bar{p}\rho).$$

The expression g is convex in γ as the second derivate is $4(\rho - 1)(\bar{p} - 1) > 0$, hence the adversary chooses the extreme point

$$\gamma = \frac{-\bar{p} + \beta\bar{p} - \rho + \bar{p}\rho}{(\rho - 1)(\bar{p} - 1)}.$$

The resulting g is concave in β as the second derivative is

$$-4 - \frac{4\bar{p}^2}{(\rho - 1)(\bar{p} - 1)} < 0.$$

Hence the algorithm would like to choose the extreme point

$$\beta = \frac{1 - \bar{p} + \bar{p}^2 - \rho + 2\bar{p}\rho - \bar{p}^2\rho}{1 - \bar{p} + \bar{p}^2 - \rho + \bar{p}\rho},$$

which is the claimed expression (16). Now g depends solely on ρ and $\bar{p}$ and is increasing in both variables. Hence the smallest ρ such that $g \geq 0$ is the root of g in ρ namely

$$\rho = \frac{-1 - \bar{p} + 2\bar{p}^2 - \bar{p}^3 + \sqrt{-3 + 6\bar{p} - 3\bar{p}^2 - 6\bar{p}^3 + 10\bar{p}^4 - 4\bar{p}^5 + \bar{p}^6}}{2(\bar{p} - 1)}, \tag{17}$$

which we would clearly like to simplify. Considering the worst upper limit, namely $\bar{p} = \rho$ the ratio simplifies to

$$\rho = \frac{1 + \sqrt{3 + 2\sqrt{5}}}{2} \approx 1.8668.$$

Case $\bar{p} \geq p^*$. In this case $\beta \leq 0$ and UTE first tests and postpones the first $1 - \gamma$ fraction of jobs (all of length $\bar{p}$) and then tests and executes the remaining γ fraction (all of length 0). Thus the n^2 dependent cost for the algorithm is

$$\textsc{Alg} = \gamma^2/2 + \bar{p}(1-\gamma)^2/2 + (1-\gamma)\gamma + (1-\gamma),$$

while the optimal cost is as in the previous case.

The ratio is at most ρ if $g \geq 0$ for

$$g := 2(\rho\textsc{Opt} - \textsc{Alg}) = \gamma^2(\bar{p} - 1)(\rho - 1) + \bar{p}(\rho - 1) + 2\gamma(\bar{p} + \rho - \bar{p}\rho) - 1,$$

where we used the factor 2 to obtain a simpler expression. The expression g is increasing in $\bar{p}$ as its derivative is $(1-\gamma)^2(\rho-1) > 0$. Therefore we can assume for the worst case $\bar{p} = p^*$. Now we observe that g is convex in γ as the second derivative is $1+\sqrt{4\rho - 3} > 0$. Hence the adversary chooses the extreme point for g in γ, namely

$$\gamma = \frac{-1 + \sqrt{4\rho - 3}}{1 + \sqrt{4\rho - 3}}.$$

With these choices of $\bar{p}$ and γ the expression g has the form

$$g = \frac{3 - 2(2 - \rho)\rho - \sqrt{4\rho - 3}}{2(\rho - 1)}.$$

Evaluated at $\frac{1+\sqrt{3+2\sqrt{5}}}{2}$ the goal is positive, proving the ratio in this case.

Case $\rho \leq \bar{p} \leq p^*$ *and* $\gamma \geq 1 - \beta$. In this case the algorithm does not postpone the execution of jobs. The jobs in the first $1 - \gamma$ fraction have processing time $\bar{p}$ and the last γ fraction jobs have processing time 0. Therefore the cost of UTE is

$$\textsc{Alg} = (\bar{p} + 1)(1 - \gamma)^2/2 + \gamma^2/2 + (\bar{p} + 1)\gamma(1 - \gamma).$$

In this case g is

$$g := 2(\rho\textsc{Opt} - \textsc{Alg}) = -1 - (1 - \gamma^2)\bar{p} + (\bar{p} - (2 - \gamma)\gamma(\bar{p} - 1))\rho.$$

The value of β is maximized at $\bar{p} = \rho$, which is approximately $\beta^* := 0.2869$. We observe that the derivative of g in $\bar{p}$ is negative in the range $\gamma \in [1 - \beta^*, 1]$, hence g is minimized at $\bar{p} = p^*$. For this choice g has the approximate form

$$1.4235 + \gamma(-6.7057 + 6.1489\gamma)$$

which can never become negative, even in the range $\gamma \in [0, 1]$. Therefore we have shown that the ratio is at most ρ also in this last case.

Analysis of the n dependent parts of the costs. Again we consider the same 3 cases as before.

Case $\rho \le \bar{p} \le p^*$ and $\gamma \le 1-\beta$: Here the n dependent costs of UTE and Opt are

$$\begin{aligned} \textsc{Alg} &= (\bar{p}+1)\beta/2 + \gamma/2 + \bar{p}(1-\beta-\gamma)/2 \\ \textsc{Opt} &= \gamma/2 + \bar{p}(1-\gamma)/2. \end{aligned}$$

The ratio is at most ρ if $g \ge 0$ for

$$g := \rho\textsc{Opt} - \textsc{Alg} \approx 0.7309 - 0.4232\gamma$$

which is positive for all $\gamma \in [0,1]$.

Case $\bar{p} \ge p^*$: The n dependent cost of UTE is

$$\textsc{Alg} = \gamma/2 + (\bar{p}+1)(1-\gamma)/2$$

leading to

$$\rho\textsc{Opt} - \textsc{Alg} \approx 0.7118 - 0.2784\gamma$$

which again is always positive.

Case $\rho \le \bar{p} \le p^*$ and $\gamma \ge 1-\beta$: This time we have

$$\textsc{Alg} = (\bar{p}+1)(1-\gamma)/2 + \gamma/2$$

and

$$\rho\textsc{Opt} - \textsc{Alg} \approx 0.3508 + 0.0768\gamma$$

which completes the proof.

□

The deterministic lower bound 1.8546 in Theorem 9 uses the upper limit $\bar{p} \approx 1.9896$. Plugging this choice of $\bar{p}$ into the expression (17) shows that UTE has a near-optimal competitive ratio.

Corollary 19 UTE *has competitive ratio $\rho \approx 1.8552$ for scheduling with testing to minimize the sum of completion times for instances with upper limits $\bar{p} \approx 1.9896$.*

6 Optimal Testing for Minimizing the Makespan

We consider scheduling with testing with the objective of minimizing the makespan, i.e., the completion time of the last job that is completed. This objective function is special, as the time each job spends on the machine has a linear contribution to the objective function value. This yields that for any algorithm that treats each job independent of the position where it occurs in the schedule, there is a worst-case instance containing only a single job.

Lemma 20 *If an algorithm that treats each job independent of the position where it occurs in the schedule is ρ-competitive for one-job instances, it is ρ-competitive also for general instances.*

Proof Let an instance I with n jobs $j_1, \ldots, j_n$ and an arbitrary algorithm as in the statement of the lemma be given. Then the makespan $ALG(I)$ equals the sum of the makespans, if we split the instance into one-job instances. By assumption, the algorithm is ρ-competitive for each one-job instance. Thus, we have

$$\textsc{Alg}(I) = \sum_{i=1}^{n} \textsc{Alg}(\{j_i\}) \leq \sum_{i=1}^{n} \rho \cdot \textsc{Opt}(\{j_i\}) = \rho \cdot \textsc{Opt}(I).$$

□

Deterministic algorithms. We apply Lemma 20 to give a deterministic algorithm with competitive ratio $\rho = \varphi$, the golden ratio, and show this is best-possible.

Theorem 21 *Let $\varphi \approx 1.618$ be the golden ratio. Testing each job j if and only if $\bar{p}_j > \varphi$ is an algorithm with competitive ratio φ for scheduling with testing to minimize the makespan. This is best possible for deterministic algorithms.*

Proof By Lemma 20 we just need to consider an instance consisting of a single job. Let that job have upper limit $\bar{p}$ and processing time p. If the algorithm does not test the job, then $\bar{p} \leq \varphi$. If $\bar{p} \leq 1$, the optimal schedule also executes the job untested, and the competitive ratio is 1. If $\bar{p} > 1$, the makespan of the algorithm is $\bar{p} \leq \varphi$ and the optimal makespan is at least 1, because the optimal makespan is minimized if the job is tested in the optimal schedule and reveals $p = 0$. Thus, the ratio is at most φ.

If the algorithm tests the job, then its makespan is $1+p$, while the optimal makespan is $\min\{\bar{p}, 1+p\}$. In the worst case, the job has processing time $p = \bar{p}$. Then the ratio is $(1+\bar{p})/\bar{p}$, which decreases when the upper limit $\bar{p}$ increases. Thus, it is at most $(1+\varphi)/\varphi = \varphi$.

To show this is best-possible, consider an instance with a single job with upper limit φ. Any algorithm that does not test this job has competitive ratio at least φ, as the optimal makespan is 1 if the job has processing time 0. Any other algorithm tests the job. If the job has processing time φ, the competitive ratio is $(1+\varphi)/\varphi = \varphi$. □

This shows that there is an algorithm that approaches the optimal execution time up to a factor φ. However, this does not lead to a φ-approximation for the problem of minimizing the sum of completion times where the additional difficulty lies in determining the job order.

Randomized algorithms. For randomized algorithms, we first show that no randomized (or deterministic) algorithm can have competitive ratio $\rho < 4/3$.

Theorem 22 *No algorithm has competitive ratio $\rho < 4/3$ for minimizing the makespan (resp. the sum of completion times) for scheduling with testing.*

Proof We want to apply Yao's principle [60] and give a randomized instance for which no deterministic algorithm is better than 4/3-competitive. Consider a one-job instance with $\bar{p} = 2$. Let the job have $p = 0$ and $p = 2$ each with probability 0.5. The deterministic algorithm that does not test the job has expected makespan 2 and the deterministic algorithm testing the job also has expected makespan 2. The expected optimal makespan is 3/2. Thus, the instance yields the desired bound. □

For minimizing the makespan the order in which jobs are treated is irrelevant by Lemma 20. Thus, the only decision an algorithm has to take is whether to test a job. Consider a job with upper limit $\bar{p}$. We show that the algorithm that executes the job untested if $\bar{p} \leq 1$ and otherwise tests it with probability $1 - 1/(\bar{p}^2 - \bar{p} + 1)$ is best-possible.

Theorem 23 *Our randomized algorithm testing each job with $\bar{p} > 1$ with probability $1-1/(\bar{p}^2-\bar{p}+1)$ has competitive ratio $4/3$ for scheduling with testing to minimize the makespan. This is best-possible.*

Proof By Lemma 20 we just need to consider an instance consisting of a single job. If its upper limit $\bar{p}$ satisfies $\bar{p} \leq 1$, the algorithm executes the job untested, which is optimal. Therefore, assume for the rest of the proof that $\bar{p} > 1$.

Note that Proposition 2, which was stated in the context of minimizing the sum of completion times, holds also for single-job instances where the objective is the makespan, because for one job the two objectives are the same. If $0 < p < \bar{p} - 1$, we observe that the optimal makespan and the expected makespan of the algorithm depend linearly on p, so by Proposition 2 we can set p to 0 or $\bar{p}-1$ without decreasing the competitive ratio. Now, if $\bar{p} - 1 \leq p < \bar{p}$, observe that increasing p to $\bar{p}$ increases the expected makespan of the algorithm but does not affect the optimum. Therefore, we can assume that $p \in \{0, \bar{p}\}$ in a worst-case instance.

Let us first consider the case $p = \bar{p}$. Then the optimal solution schedules this job without test. Thus, the ratio of algorithm length over optimal length is

$$\rho = \frac{E[\textsc{Alg}]}{\textsc{Opt}} = \left(1 - \frac{1}{\bar{p}^2 - \bar{p} + 1}\right) \frac{\bar{p}+1}{\bar{p}} + \frac{1}{\bar{p}^2 - \bar{p} + 1} = \frac{\bar{p}^2}{\bar{p}^2 - \bar{p} + 1}.$$

Otherwise, we have $p = 0$. Then we have

$$\rho = \frac{E[\textsc{Alg}]}{\textsc{Opt}} = \left(1 - \frac{1}{\bar{p}^2 - \bar{p} + 1}\right) + \frac{1}{\bar{p}^2 - \bar{p} + 1}\bar{p} = \frac{\bar{p}^2}{\bar{p}^2 - \bar{p} + 1}.$$

This function is maximized at $\bar{p} = 2$, which yields the competitive ratio $4/3$. □

7 Conclusion

In this paper we have introduced an adversarial model of scheduling with testing where a test can shorten a job but the time for the test also prolongs the schedule, thus making it difficult for an algorithm to find the right balance between tests and executions. We have presented upper and lower bounds on the competitive ratio of deterministic and randomized algorithms for a single-machine scheduling problem with the objective of minimizing the sum of completion times or the makespan. An immediate open question is whether it is possible to achieve competitive ratio below 2 for minimizing the sum of completion times with a deterministic algorithm for arbitrary instances. Further interesting directions for future work include the consideration of job-dependent test times or other scheduling problems such as parallel machine scheduling or flow shop problems. More generally, the study of problems with explorable uncertainty in settings where the costs for querying uncertain data directly contribute to the objective value is a promising direction for future work.

Acknowledgements We would like to thank Markus Jablonka and Bruno Gaujal for helpful discussions about the algorithm DelayAll, as well as an anonymous referee for pointing us to related work on exploration versus exploitation in the multi-armed bandit framework.

References

1. M. Adamczyk, M. Sviridenko, and J. Ward. Submodular stochastic probing on matroids. *Mathematics of Operations Research*, 41(3):1022–1038, 2016.
2. S. Alizamir, F. de Véricourt, and P. Sun. Diagnostic accuracy under congestion. *Management Science*, 59(1):157–171, 2013.
3. S. Assadi, S. Khanna, and Y. Li. The stochastic matching problem with (very) few queries. *ACM Trans. Economics and Comput.*, 7(3):16:1–16:19, 2019.
4. N. Bansal, A. Gupta, J. Li, J. Mestre, V. Nagarajan, and A. Rudra. When LP is the cure for your matching woes: Improved bounds for stochastic matchings. *Algorithmica*, 63(4):733–762, 2012.
5. N. Bansal and V. Nagarajan. On the adaptivity gap of stochastic orienteering. *Math. Program.*, 154(1-2):145–172, 2015.
6. S. Behnezhad, A. Farhadi, M. Hajiaghayi, and N. Reyhani. Stochastic matching with few queries: new algorithms and tools. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2855–2874. SIAM, 2019.
7. A. Blum, J. P. Dickerson, N. Haghtalab, A. D. Procaccia, T. Sandholm, and A. Sharma. Ignorance is almost bliss: Near-optimal stochastic matching with few queries. *Operations Research*, 68(1):16–34, 2020.
8. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
9. S. Bubeck and N. Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1):1–122, 2012.
10. J. M. P. Cardoso, J. G. de Figueiredo Coutinho, and P. C. Diniz. *Embedded Computing for High Performance: Efficient Mapping of Computations Using Customization, Code Transformations and Compilation*. Morgan Kaufmann, 2017.
11. N. Chen, N. Immorlica, A. R. Karlin, M. Mahdian, and A. Rudra. Approximating matches made in heaven. In *36th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 5555 of *Lecture Notes in Computer Science*, pages 266–278. Springer, 2009.
12. C.-F. M. Chou, M. Queyranne, and D. Simchi-Levi. The asymptotic performance ratio of an on-line algorithm for uniform parallel machine scheduling with release dates. *Mathematical Programming*, 106(1):137–157, 2006.
13. B. C. Dean, M. X. Goemans, and J. Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. *Mathematics of Operations Research*, 33(4):945–964, 2008.
14. E. Demeulemeester and W. Herroelen. Robust project scheduling. *Foundations and Trends in Technology, Information and Operations Management*, 3(3-4):201–376, 2010.
15. I. Dumitriu, P. Tetali, and P. Winkler. On playing golf with two balls. *SIAM J. Discret. Math.*, 16(4):604–615, 2003.
16. C. Dürr, T. Erlebach, N. Megow, and J. Meißner. Scheduling with explorable uncertainty. In A. R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 94 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
17. T. Erlebach, M. Hoffmann, D. Krizanc, M. Mihalák, and R. Raman. Computing minimum spanning trees with uncertainty. In S. Albers and P. Weil, editors, *25th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1 of *LIPIcs*, pages 277–288. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2008.
18. T. Feder, R. Motwani, L. O'Callaghan, C. Olston, and R. Panigrahy. Computing shortest paths with uncertainty. *Journal of Algorithms*, 62(1):1–18, 2007.
19. T. Feder, R. Motwani, R. Panigrahy, C. Olston, and J. Widom. Computing the median with uncertainty. *SIAM Journal on Computing*, 32(2):538–547, 2003.
20. A. Ferber, M. Krivelevich, B. Sudakov, and P. Vieira. Finding hamilton cycles in random graphs with few queries. *Random Struct. Algorithms*, 49(4):635–668, 2016.

21. A. Ferber, M. Krivelevich, B. Sudakov, and P. Vieira. Finding paths in sparse random graphs requires many queries. *Random Struct. Algorithms*, 50(1):71–85, 2017.
22. A. Fiat and G. J. Woeginger, editors. *Online Algorithms: The State of the Art*, volume 1442 of *LNCS*. Springer, 1998.
23. J. Focke, N. Megow, and J. Meißner. Minimum spanning tree under explorable uncertainty in theory and experiments. In *16th International Symposium on Experimental Algorithms (SEA)*, volume 75 of *LIPIcs*, pages 22:1–22:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.
24. J. Gittins, K. Glazebrook, and R. Weber. *Multi-armed Bandit Allocation Indices*. Wiley, 2nd edition, 2011.
25. J. C. Gittins. A dynamic allocation index for the sequential design of experiments. *Progress in statistics*, pages 241–266, 1974.
26. M. Goerigk, M. Gupta, J. Ide, A. Schöbel, and S. Sen. The robust knapsack problem with queries. *Computers & OR*, 55:12–22, 2015.
27. A. Gupta, H. Jiang, Z. Scully, and S. Singla. The Markovian price of information. In *International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, volume 11480 of *Lecture Notes in Computer Science*, pages 233–246. Springer, 2019.
28. A. Gupta, R. Krishnaswamy, V. Nagarajan, and R. Ravi. Running errands in time: Approximation algorithms for stochastic orienteering. *Mathematics of Operations Research*, 40(1):56–79, 2015.
29. A. Gupta and V. Nagarajan. A stochastic probing problem with applications. In *nternational Conference on Integer Programming and Combinatorial Optimization (IPCO)*, volume 7801 of *Lecture Notes in Computer Science*, pages 205–216. Springer, 2013.
30. A. Gupta, V. Nagarajan, and S. Singla. Algorithms and adaptivity gaps for stochastic probing. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 1731–1747. SIAM, 2016.
31. M. Gupta, Y. Sabharwal, and S. Sen. The update complexity of selection and related problems. *Theory of Computing Systems*, 59(1):112–132, 2016.
32. S. Kahan. A model for data in motion. In *23rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 267–277, 1991.
33. A. Kasperski and P. Zieliński. Risk-averse single machine scheduling: complexity and approximation. *Journal of Scheduling*, 22(5):567–580, 2019.
34. P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45, 2019.
35. S. Khanna and W.-C. Tan. On computing functions with uncertainty. In *20th Symposium on Principles of Database Systems (PODS)*, pages 171–182, 2001.
36. R. D. Kleinberg, B. Waggoner, and E. G. Weyl. Descending price optimally coordinates search. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, pages 23–24. ACM, 2016.
37. P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications*. Springer, 1997.
38. J. Y.-T. Leung. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman & Hall/CRC, 2004.
39. R. Levi, T. L. Magnanti, and Y. Shaposhnik. Scheduling with testing. *Management Science*, 65(2):776–793, 2019.
40. W. Ma. Improvements and generalizations of stochastic knapsack and markovian bandits approximation algorithms. *Mathematics of Operations Research*, 43(3):789–812, 2018.
41. T. Maehara and Y. Yamaguchi. Stochastic packing integer programs with few queries. *Mathematical Programming*, pages 1–34, 2019.
42. S. Marbán, C. Rutten, and T. Vredeveld. Learning in stochastic machine scheduling. In *9th International Workshop on Approximation and Online Algorithms (WAOA)*, volume 7164 of *Lecture Notes in Computer Science*, pages 21–34. Springer, 2011.
43. N. Megow, J. Meißner, and M. Skutella. Randomization helps computing a minimum spanning tree under uncertainty. *SIAM Journal on Computing*, 46(4):1217–1240, 2017.
44. N. Megow, M. Uetz, and T. Vredeveld. Models and algorithms for stochastic online scheduling. *Mathematics of Operations Research*, 31(3):513–525, 2006.
45. N. Megow and T. Vredeveld. A tight 2-approximation for preemptive stochastic scheduling. *Mathematics of Operations Research*, 39(4):1297–1310, 2014.
46. A. F. Mills, N. T. Argon, and S. Ziya. Resource-based patient prioritization in mass-casualty incidents. *Manufacturing & Service Operations Management*, 15(3):361–377, 2013.

47. R. Möhring, F. Radermacher, and G. Weiss. Stochastic scheduling problems I: General strategies. *Zeitschrift für Operations Research*, 28:193–260, 1984.
48. R. Möhring, A. Schulz, and M. Uetz. Approximation in stochastic scheduling: The power of LP-based priority policies. *Journal of the ACM*, 46:924–942, 1999.
49. R. P. Nicolai and R. Dekker. *Optimal Maintenance of Multi-component Systems: A Review*, pages 263–286. Springer London, 2008.
50. C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *26th International Conference on Very Large Data Bases (VLDB)*, pages 144–155, 2000.
51. W. P. Pierskalla and J. A. Voelker. A survey of maintenance models: The control and surveillance of deteriorating systems. *Naval Research Logistics*, 23(3):353––388, 1976.
52. M. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Springer Science+Business Media, fifth edition, 2016.
53. K. Pruhs, J. Sgall, and E. Torng. Online scheduling. In J. Y.-T. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 15. Chapman & Hall/CRC, 2004.
54. J. A. Rothstein. Adaptive compression, July 30 2013. US Patent 8,499,100.
55. Y. Shaposhnik. *Exploration vs. Exploitation: Reducing Uncertainty in Operational Problems.* PhD thesis, Sloan School of Management, MIT, 2016.
56. S. Singla. The price of information in combinatorial optimization. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2523–2532. SIAM, 2018.
57. W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.
58. M. Weitzman. Optimal search for the best alternative. *Econometrica*, 47(3):641–54, 1979.
59. Y. Wiseman, K. Schwan, and P. Widener. Efficient end to end data exchange using configurable compression. *ACM SIGOPS Operating Systems Review*, 39(3):4–23, 2005.
60. A. C.-C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 222–227. IEEE, 1977.