Technische Universität München
Fakultät für Informatik

Online Algorithms for Scheduling with Testing

Alexander Eckl

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzende: Prof. Dr. Stefanie Rinderle-Ma

Prüfende der Dissertation:

1. Prof. Dr. Susanne Albers
2. Prof. Dr. Andreas Schulz

Die Dissertation wurde am 14.09.2021 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 15.02.2022 angenommen.

Abstract

Scheduling with testing is a recent online optimization problem in the framework of explorable uncertainty which models situations where some preliminary action can decrease the duration of tasks. The problem is motivated by real-world applications including production planning, market demand predictions, distributed computing, and medical diagnoses. A given number of jobs with uncertain processing times has to be assigned to a set of given machines. Jobs can be run in one of two possible configurations: Running a job *tested* reveals the previously unknown processing time, and the job can then be executed on some machine. Running the job *untested* takes a predetermined amount of time which is always at least as large the job's actual processing time. The difficulty of the scheduling with testing problem lies in balancing the uncertain but possibly beneficial investment into the test and the fallback strategy of running the job untested with a fixed duration.

We study the problem on a single machine as well as on multiple identical machines and examine the objectives of minimizing the makespan and the total completion time of the schedule. We also differentiate between non-preemptive, test-preemptive, and preemptive settings for both objectives. We provide several new, theoretical algorithms and lower bounds for the problem and study their performance based on the concept of competitive analysis. For this, we utilize testing schemes based on job parameters, pairwise examination of contributions to the completion time of jobs, and elaborate sorting procedures. We also adapt and extend well-known algorithms from the literature, like List Scheduling and Round Robin, to our problem.

For the makespan objective, we prove optimal algorithms and lower bounds on a single machine, both for the deterministic and the randomized case. On multiple machines, we present several algorithms with constant competitive ratios for the non-preemptive setting, which are complemented by a lower bound with constant value. In addition, we give an algorithm and a lower bound which apply to both test-preemptive and preemptive settings and are optimal if the number of machines approaches infinity. We also consider an extension of the problem to sequential online arrivals of jobs, for which we provide improved lower bounds. For the sum of completion times, we examine constant-competitive algorithms for the test-preemptive and the preemptive setting on one machine and show that such a result cannot exist in the non-preemptive variant. Additionally, we present a lower bound for the preemptive setting as well as an improved result for randomized algorithms.

Zusammenfassung

Scheduling with Testing ist ein aktuelles Online-Optimierungsproblem im Rahmen von erkundbarer Unsicherheit, welches Situationen modelliert, bei denen eine vorangehende Maßnahme die Dauer von Tätigkeiten reduzieren kann. Das Problem wird motiviert durch reale Anwendungen, wie zum Beispiel Produktionsplanung, Prognosen über Marktnachfrage, verteilte Rechensysteme, oder medizinische Diagnosen. Eine gegebene Anzahl von Aufgaben mit unsicheren Laufzeiten soll einer Menge von gegebenen Maschinen zugewiesen werden. Aufgaben können in einer von zwei Konfigurationen ausgeführt werden: Wird die Aufgabe *getestet* ausgeführt, so wird die bisher unbekannte Laufzeit aufgedeckt, und die Aufgabe kann anschließend auf einer Maschine ausgeführt werden. Eine Aufgabe *ungetestet* auszuführen dauert eine vorher festgelegte Zeitspanne, welche immer mindestens so groß ist wie die eigentliche Laufzeit des Jobs. Die Schwierigkeit des Scheduling with Testing-Problems liegt in der Balance zwischen der unsicheren aber möglicherweise vorteilhaften Investition in den Test und der Rückfallstrategie die Aufgabe ungetestet mit feststehender Dauer auszuführen.

Wir untersuchen das Problem auf einer einzelnen Maschine sowie auf mehreren identischen Maschinen und betrachten als Zielsetzung die Minimierung der Produktionsspanne und der Summe der Komplettierungszeiten des Zeitplans. Wir unterscheiden außerdem zwischen nicht-präemptiven, Test-präemptiven, und präemptiven Szenarien für beide Zielfunktionen. Wir stellen mehrere neue, theoretische Algorithmen und untere Schranken für das Problem bereit und untersuchen deren Leistungen basierend auf dem Konzept der kompetitiven Analyse. Dafür benutzen wir Test-Schemata beruhend auf Aufgabenparametern, die paarweise Auswertung von Beiträgen zur Komplettierungszeit von Aufgaben, und ausführliche Sortierungsmethoden. Außerdem adaptieren und erweitern wir bekannte Algorithmen aus der Literatur, wie List Scheduling und Round Robin, auf unser Problem.

Für die Zielfunktion der Produktionsspanne beweisen wir optimale Algorithmen und untere Schranken auf einer einzelnen Maschine, sowohl für den deterministischen als auch den randomisierten Fall. Auf mehreren Maschinen präsentieren wir mehrere Algorithmen mit konstantem kompetitivem Verhältnis für das nicht-präemptive Szenario, welche durch eine untere Schranke mit konstantem Wert ergänzt werden. Weiterhin stellen wir einen Algorithmus und eine untere Schranke vor, die sowohl für Test-präemptive als auch präemptive Szenarien angewandt werden können und optimal sind, wenn die Anzahl an Maschinen gegen unendlich geht. Wir betrachten zusätzlich eine Erweiterung des Problems auf sequenzielle Online-Ankünfte von Aufgaben, wofür wir verbesserte untere Schranken bereitstellen. Für die Summe der Komplettierungszeiten untersuchen wir konstant kompetitive Algorithmen für das Test-präemptive und das präemptive Szenario auf einer Maschine und zeigen, dass ein solches Resultat in der nicht-präemptiven Variante nicht möglich ist. Außerdem präsentieren wir eine untere Schranke für das präemptive Szenario und ein verbessertes Resultat für randomisierte Algorithmen.

Acknowledgments

This work would not have been possible without the help and effort of many people: my supervisors, colleagues, co-authors, friends, and family. I owe it to all of you that I was able to pursue my doctorate and publish this thesis, and I would like to extend my utmost gratitude.

First and foremost, I would like to express my gratitude to my supervisor, Prof. Dr. Susanne Albers, for her guidance, trust, and advice during my time as a doctoral candidate at the Technical University of Munich. I am equally thankful to my second supervisor, Prof. Dr. Maximilian Schiffer, for his great support and advice, and for always making time for a meeting, even in his busy schedule. In addition, I want to thank my mentor Prof. Dr. Peter Gritzmann and the other members of the examination committee, Prof. Dr. Stefanie Rinderle-Ma and Prof. Dr. Andreas Schulz.

Many thanks to all my present and former colleagues at the chair and at AdONE: Giacomo Dall'Olio, Yoshimi von Felbert, Alexandre Forel, Stefan Kober, Isabel Koch, Leon Ladewig, Richard Littmann, Maximilian Janke, Layla Martin, Jens Quedenfeld, Richard Stotz, Stefan Weltge, and many other amazing people. It was great to work with you all and I enjoyed the time we spent together during and outside of work immensely. Special thanks go to my co-authors Anja Kirschbaum, Marilena Leichter, Luisa Peter, and Kevin Schewior. Thank you for the excellent collaboration on our papers and all the long hours we spent together exchanging ideas, writing down results, or despairing at proofs, while never losing sight of the funny side of things.

I am particularly indebted to the people who have helped proofread parts of this thesis: Martin Ha Minh, Martin Bullinger, Stefan Kober, Waldo Gálvez, and Maximilian Janke. Because of your hard work and input, the quality of this text has improved greatly.

Finally, my deepest gratitude and thanks go out to all of my friends and family, who supported and endured me through these last few years. Thank you for all the fun we had and the great moments we experienced together. To my girlfriend Lisa: I love you to the ends of this world. My final Thank You goes to my parents for their unlimited love and support and for always being there for me no matter the circumstances.

Contents

1 Introduction

In the fundamental research area of scheduling the goal is to efficiently distribute, allocate, or sort limited resources to complete jobs over given time periods. As such it is being used widely in industries like manufacturing, service provision, IT management, and resource planning, to name just a few. Commonly, a single decision maker has to find a strategy to optimize some kind of cost or reward function based on the schedule, for example minimizing production times or maximizing revenue. At the same time they must adhere to numerous constraints that may overlap each other, like bounded resources or personnel restrictions. The performance of a given approach can be evaluated for example by mathematical analysis or experimental investigation.

As part of the theoretical field of operations research, scheduling has been one of the most established and fruitful settings in the last decades — with substantial advances since its inception during the 1950s [74]. Earliest work for scheduling as an independent area includes seminal papers by Johnson [58] and Smith [83]. Today, scheduling remains one of the most widely studied areas in operations research. The fundamental challenges of a modern, networked economy continually require new and up-to-date methods for an ever-increasing landscape of problems.

In this thesis, we mathematically design and analyze algorithms for a recent scheduling problem called *scheduling with testing*. It arises from real-life applications where partially unknown information about tasks can be obtained at some given additional cost. As such, the problem involves *uncertainty* with respect to the input values that are used by the decision maker to define a strategy.

Scheduling settings with uncertainty are characterized by some form of incomplete information about the problem input. Possible reasons for uncertain input are plentiful: missing or unobtainable data, information that changes depending on time or context, or simply the difficulty of assessing the available information properly. Different frameworks have been used in the operations research literature to handle uncertainty [62]: *Stochastic optimization* assumes that the unknown input behaves based on a known random distribution. Dantzig [31] studied this setting already in 1955. In *robust optimization*, the worst possible outcome over all inputs is analyzed. Earliest work in this area includes Soyster [84]. In this thesis, we study *online optimization*, where uncertain information is presented sequentially over time to the problem solver. Online optimization settings in scheduling were investigated by Graham [51] as early as 1966.

In the comparatively new scheduling with testing problem, a given number of jobs whose processing times are unknown have to be assigned to a given set of machines such that certain objective functions based on the completion time of the jobs are minimized. The characteristic feature of this setting is that the scheduler can invest additional resources to *test* a job and reveal its processing length. The cost of testing is

added directly to one of the machines responsible for running the jobs. Alternatively, a job can be executed without testing, a kind of fallback strategy which takes a fixed amount of time that is at least as long as the true processing time of the job. Since additional information is revealed during the runtime of an algorithm, this problem can be interpreted as part of the online optimization framework.

In online optimization, unknown input values are made available at fixed times during the runtime of an algorithm, potentially depending on previous choices it has already made. A very common way to mathematically analyze such *online algorithms* is the concept of *competitive analysis*, first considered by Sleator and Tarjan [82], where an algorithm is compared against a best possible offline strategy with complete information. The *competitive ratio* is defined as the worst-case ratio over all inputs between the value of the algorithm and the value of the optimal offline solution [23].

This thesis addresses the following research questions for the topic of scheduling with testing: How can we develop new algorithms to solve the problem theoretically such that solutions lie within a provably good factor of the best possible solution as defined by the concept of the competitive ratio? Can we construct explicit examples such that a *specific* algorithm performs poorly on these instances, yielding an impossibility result for the given algorithm? Finally, which theoretical lower bounds can we provide in the competitive analysis framework such that it is provably impossible for *any* algorithm to achieve a ratio smaller than the given bound?

1.1 Scheduling with Testing

In many applications arising in IT management, resource planning, or manufacturing, partially or completely uncertain information is found. To compensate for these uncertainties, some preparatory action is often taken to gain more information: A construction firm usually prepares cost estimates for their clients before they start building a house. Similarly, a medical practitioner may run several elaborate examinations before deciding on a diagnosis.

Whenever there is the possibility of gaining more precise information through some preliminary action, usually involving some additional cost, we can model this situation mathematically using the *scheduling with testing* framework. The process of investing additional resources like time, computing power, human labor, or money, to attain the previously unknown information is referred to as *testing* in this model.

Additionally, we assume that there is always a fallback strategy that can be executed whenever we do not want to invest these additional resources. For example, instead of paying the uncertain repair cost for an old and damaged car, we may invest our money into buying a new car. The cost of this *upper bound* strategy is always assumed to be at least as large as the resulting cost of the job if we had invested into the test. There is a natural trade-off in the model between the uncertain but possibly beneficial investment of resources and the upper bound strategy with known cost.

Mathematically, we consider a set of m machines, where n jobs have to be assigned. All job processing times $p_j \geq 0$ are unknown. However, the algorithm is given known

upper bounds u_j which fulfill $u_j \geq p_j$, as well as known testing times $t_j \geq 0$. The algorithm can decide for each job in which configuration it should be executed on a machine: Testing and running its processing time takes a total time of $t_j + p_j$, while executing it untested takes time equal to u_j. The true processing time p_j of a job is only revealed if the job is run in the tested configuration, and then only after completing the test. A rigorous formal definition of the mathematical model is given in Chapter 2.

The scheduling with testing problem has first been introduced in this form by Dürr et al. [35] in 2018. They consider the single machine case and give results for minimizing the sum of completion times and the makespan.

The problem falls into the framework of *explorable uncertainty*, a field concerned with optimizing under uncertain parameters that can be explored by the algorithm for a certain cost. Explorable uncertainty has been considered in previous research extensively, see for example [24, 38, 59, 81].

The model is also naturally connected with classic machine scheduling models on a single machine, as well as on multiple machines. We mention in particular the sum of completion times objective on a single machine [1,66,78,83], and makespan minimization on multiple machines [46, 51, 76], since these settings are primarily considered in this thesis. For a first overview, we recommend the book on scheduling by Pinedo [73]. For more details on previous work and related literature, we refer to Chapter 3.

Compared to some of the traditional machine scheduling settings, the possibility of testing jobs to improve their processing by an a priori unknown amount increases the complexity of the scheduling with testing problem. Even for a single job, an algorithm must make the non-trivial decision to invest into gaining more information or using the fallback strategy. Additional decisions concerning the order of the jobs in the sum of completion times objective or the assignment of jobs to multiple machines add even more intricacies to the problem.

To fulfill the tasks determined by our research questions, we present and analyze several deterministic online algorithms as well as lower bounds for the scheduling with testing setting in this thesis. We consider one or more machines for both the objective of minimizing the sum of completion times as well as the makespan. In our analysis for a single machine, we utilize both established and new testing rules based on job parameters, introduce tailored sorting schemes, and evaluate pairwise contributions of jobs to their completion times. For more than one machine, we extend the testing rules from the single machine case, sort jobs into groups based on their parameters, and extend classic proof methods from makespan minimization. We consider different settings concerning *preemption*, where jobs may be interrupted at predefined times during their runtime. Finally, we also consider two *randomized* algorithms in the single machine case. Our results and methods are presented in Chapters 4 and 5, respectively.

1.2 Motivating Examples and Applications

Scheduling with testing has many direct applications in real-world settings, including data and resource management, manufacturing, construction and maintenance work,

market demand predictions, distributed computing, and medical diagnoses. In this section, we talk about several such applications and consider a number of explicit examples in detail.

Consider first a very simple example in car maintenance. Let us assume the engine of our car does not start anymore as a consequence of a defective battery. We might decide to bring the car to a repair shop and simply exchange the battery, inducing a fixed cost for the new battery and the workers' hours. Alternatively, we can instruct the shop to first run a fault diagnosis, which also incurs a fixed, but smaller cost. In the worst case, the battery is in fact faulty and has to be exchanged, and we have to pay for a new one in addition to the cost of the diagnosis. But if it turns out that the cause of the problem was just an electronic malfunction that drained the fully operational battery, it might be enough to fix the malfunction and recharge the battery, saving a lot of money. In this example, we have a simple trade-off for a single task that might become cheaper by running a preliminary diagnosis, illustrating a basic instance of our problem with one job.

Next, we examine a setting in market demand predictions. Say, a production company conducts online surveys to determine customer demands for newly developed products. The results of a survey can be used to accurately predict necessary production volumes to satisfy the demand, likely leading to reduced production costs. However, conducting the survey beforehand takes a certain amount of time and money. Thus, the company has to decide whether to run a survey for a given product or not. In addition, this example highlights the need for the possibility of *preemption* in our model: After the responsible department has finished evaluating the survey for a product, it is unlikely that the same people immediately start working on its production. Instead, this is managed by an entirely different department later on, which we can model in the scheduling with testing problem by interrupting the associated job and resuming it later on a different machine.

In order to investigate a more complex example, we look at a distributed computing setting with a number of remote computing servers and one centralized master server. Data is exchanged between the master and the remote servers. Over time, the saved data at the remote servers may become out of date or inconsistent with the master. Many distributed applications allow this kind of inconsistency, often called *stale replication*, since keeping data consistent all the time is infeasible from a performance perspective [71]. However, using replicated data for computing jobs at the local servers may lead to results with lower quality. At the same time, querying the master server for the precise data incurs a query cost, e.g. the waiting time for the answer from the master. This trade-off can be modeled directly in the scheduling with testing setting by letting the cost for querying the master correspond to the execution of a test. The goal is then to minimize quality loss of the computing jobs and the total query duration or costs. In the scheduling with testing model, both objectives are combined into one. This setting features *non-uniform testing* requirements: the time to query the master server may differ substantially depending on the connection distance between the two servers. Additionally, the distributed computing framework represents an instance with multiple remote servers executing jobs, which necessitates *multiple machines* in

the scheduling with testing model.

Olston and Widom [71] provide a simple explicit example for a stale replication system: Web browsers commonly save copies of websites in the local cache of a computer. The saved copy might become out of date after some time has passed, reducing the quality of the displayed website, but refreshing a site by retrieving the master copy from a web server will take additional time, thus reducing performance.

To list some further possible scenarios, we mention manufacturing schedules where personnel has to be assigned to production tasks with uncertain duration, data transmission requests using file compression which may vary in size, or acquiring a house through an agent who gives us more information about its value, location, and condition in exchange for a fee.

In general, any circumstances where the outcome is influenced by some preceding action, be it cost or duration estimates, the acquisition of previously unknown information, or the improvement of some procedure by an indeterminate amount, can be modeled by the scheduling with testing framework.

A Detailed Example with Explicit Values. To conclude this section, we take a look at an example with explicit values to further motivate our setting and explain the difficulty as well as the distinctive traits of the mathematical model. The following example is based on running computer programs with an additional pre-optimizer that can be used to potentially improve the runtime of the code.

Suppose that we are working in a software company. One day in the morning, we receive an e-mail from our boss that the only thing we have to do on that particular day is to run a number of computing tasks. As soon as they are all finished, we are allowed to go home. All tasks we received have a known runtime and can be executed on one of three available computers. In addition, our boss also sends us a black-box optimizer which can be used to improve the runtime of any job by an unknown amount. We have no further information on how well the optimizer performs and after optimizing, the job's runtime can be anything between the initial runtime and zero. Running the optimizer takes a known, predetermined amount of time. In our simplified world, any computer running a job or the optimizer is not available to do anything else. Our goal, of course, is to go home as early as possible.

In the scheduling with testing problem, we can model the known initial runtime of a computing task as its *upper bound*, and the optimizer as the *test* that reveals the improved processing time. The computers are the machines where jobs can be assigned to. The goal of going home as early as possible can be equated to minimizing the time when the final job finishes on the last computer. This corresponds directly to the objective of minimizing the makespan of the machines.

Assume for now that our boss only sends a single computing task. The job she sends has an initial runtime of 2 hours, and the black-box optimizer runs for 1 hour. The immediate question is of course whether we should run the optimizer or not. If it is successful in decreasing the initial runtime by more than 1 hour, we get a better result, otherwise the total duration might even be larger than before.

To better understand this decision, let us fix the optimizer duration to 1 hour and consider what happens if the initial runtime of the task takes on other values larger or smaller than 2. Let us pay particular attention to the ratio between our solution and the best possible result if we knew how much the optimizer improves the runtime — this ratio is the basis of the competitive analysis framework we will use later to measure the performance of our methods.

If the initial runtime is less than 1, we speak of a *trivial* job. It makes no sense to run the optimizer, since the initial runtime of the task is actually less than the time the optimizer needs. For initial runtimes that are at least 1 but still small, it is intuitive to still not use the optimizer: Even though there is a chance that the optimizer reduces the total duration, the improvement would be comparatively small, and there is a risk of almost doubling the total duration if the optimizer does not reduce the runtime at all. In contrast, it is clear that optimizing becomes more and more lucrative the larger the initial runtime is: The comparatively small testing period in the beginning makes almost no difference to the total duration if the optimizer performs poorly, but it might possibly improve it by a large margin if the optimizer performs well.

Looking at the cases above, it becomes apparent quickly that if we have no further information about the performance of our black-box optimizer, the optimal strategy must be some kind of threshold algorithm. If the ratio between initial runtime and optimizing time is small, we shouldn't run the optimizer. If the ratio is large however, our result can improve a lot if we do. The point where we should switch between the two strategies is somewhere in between.

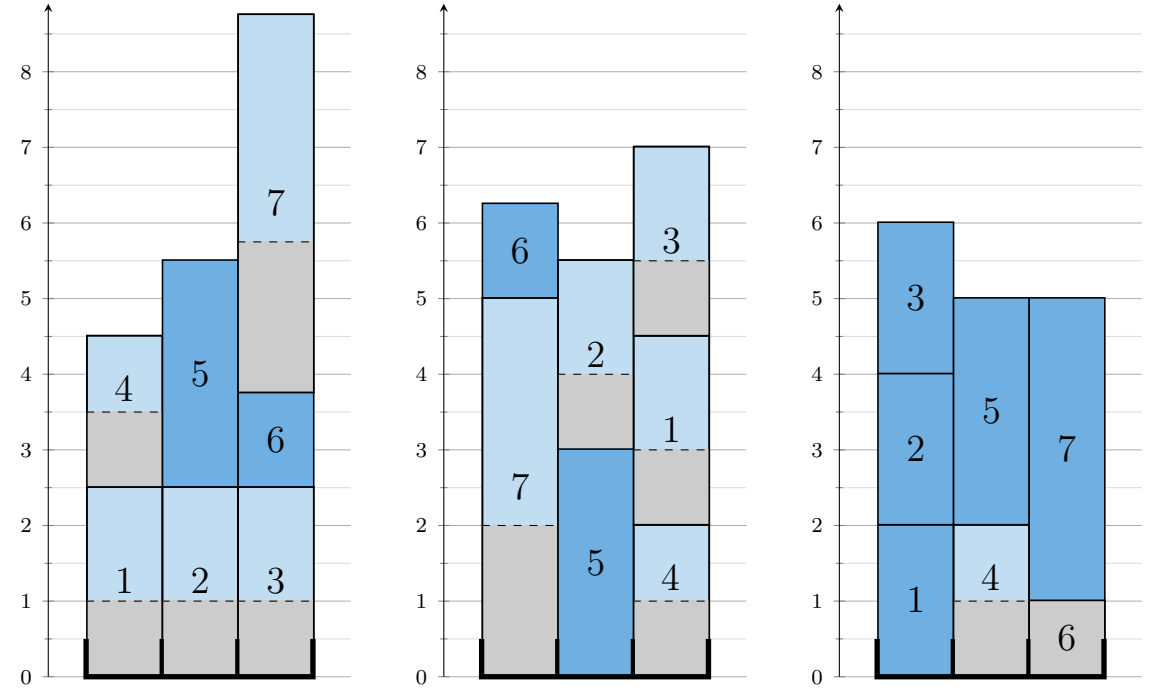
It turns out that the point where the optimal strategy switches between running the optimizer and ignoring it is approximately at a ratio of 1.6180 between initial runtime and optimizing time, a value which corresponds to the *golden ratio*. This was proven for the case when the testing time is equal to 1 by Dürr et al. [35] in their introductory paper on scheduling with testing. In [6] we generalized this result to any non-negative testing time. In Section 5 we provide more details on how to prove this result.

Let us now look at our original example and assume that the actual set of tasks sent by our boss contains more computing jobs. We give some example runtimes and the time needed by the optimizer in Table 1. The improved time of a job after optimizing is displayed in parentheses. Only when we decide to actually run the optimizer, these values are revealed to us. In particular, we have to decide whether to optimize or not without knowing them.

Now that we at least have an idea of how a single job behaves, how should we *assign* multiple jobs to our three different available computers? The *List Scheduling* algorithm

Table 1: Example parameters for a set of seven jobs.

Job number	1	2	3	4	5	6	7
Initial runtime	2	2	2	3	3	1.25	4
Optimizer duration	1	1	1	1	2	1	2
Runtime after optimizing	(1.5)	(1.5)	(1.5)	(1)	(1)	(0)	(3)



(a) List Scheduling ordered by job number

(b) List Scheduling ordered by initial runtime

(c) Optimal offline solution

Figure 1: Resulting schedules for the example instance in Table 1 based on different approaches. A job's height corresponds to its duration. Non-optimized jobs are dark blue and optimized jobs are light blue. The duration of the optimizer is indicated by the light gray area for each optimized job.

is a well-known heuristic approach to this problem. It simply assigns any job to the computer with the current smallest sum of runtimes. Our first try based on this idea is depicted in Figure 1(a): We simply consider the jobs in the given order, and use List Scheduling to assign them. Once a computer is fixed for a job, we use the optimal threshold method from above to decide whether we optimize or not. We only find out how much the runtime can be improved for those jobs which we use the optimizer on. We can see that after 8.75 hours all jobs are done and we can go home.

Can we do better than this? One advantage that we have not used in the simple approach above is that we can sort the jobs based on their known parameters. The schedule in Figure 1(b) is obtained by sorting the jobs based on non-increasing initial runtime, again using List Scheduling to decide the computers and the optimal threshold method to decide the optimizer strategy. As we can see, the time needed to finish all jobs has improved to 7 hours.

The approaches for one of the most important settings we analyze in paper [7] are based on the ideas shown in the above example. Our extension of the List Scheduling algorithm works almost exactly as shown in Figure 1(a). Our main algorithm uses a more sophisticated sorting scheme than the one used in Figure 1(b), and additionally

modifies the threshold idea to be dependent on the number of available machines.

Finally, the best possible result that can be achieved is depicted in Figure 1(c). After only 6 hours of executing tasks on our three computers we are allowed to go home, saving almost three hours compared to our first intuitive approach. This is the value against which we compare all our solutions. However, this best possible result makes heavy use of the knowledge of how much the optimizer reduces the runtime of our jobs and very often can be shown to be unattainable without this extra information. As a matter of fact, one partial goal in our analysis will be to prove that algorithms cannot achieve results better than a certain value unless they have access to this unknown information.

More details on our methods are presented in Chapter 5. One goal that we considered in paper [6] which was not described in this example, was to minimize the sum of completion times on a single machine. A brief overview of all our results, including ones not described here, can be found in the next section.

1.3 Brief Summary of Achieved Contributions

In this section we give a short high-level summary of the most important contributions of this thesis. All results were achieved in the papers [6] and [7]. All technical terms and notations used here are described in detail in Chapter 2.

We examine new algorithms and general lower bounds for the makespan and sum of completion times objectives, answering our two most important research questions to satisfaction. Additionally, some lower bounds that only hold for our given algorithms are studied. We will briefly describe the ideas and concepts behind most of the results.

Results for the Makespan. For one machine, we provide optimal algorithms and lower bounds. The deterministic ratio is $\varphi \approx 1.6180$ and the randomized ratio is $4/3$. Both results are based on a threshold strategy and extend the ratios by Dürr et al. [35], which were provided for a less general form of the problem where the testing times of all jobs are equal.

For multiple machines, we are the first to analyze algorithms and lower bounds in the setting of scheduling with testing. We examine three different deterministic algorithms whose competitive ratios increase together with the number of machines. For general testing times, we analyze a simple combination of the threshold method and List Scheduling. It has a competitive ratio of $2\varphi \approx 3.2361$ in the worst case, and there exist examples where the algorithm has the same value as its competitive ratio. By sorting the jobs into phases based on job parameters, we achieve an improved algorithm where the competitive ratio is at most 3.1016. By employing the same ideas for uniform testing times, the resulting ratio is not worse than 3. A corresponding lower bound has a value of 2 if the number of machines becomes large. It is based on a common lower bound construction for ordinary makespan minimization. Finally, for the special case when most jobs are trivial, i.e. their optimal processing time is known, we demonstrate that a uniform algorithm can have a worst-case ratio of 2.1574.

For the case when preemption is allowed, we give a straightforward algorithm with competitive ratio 2, which works by applying a brute force offline algorithm two times. The corresponding lower bound also has a value of 2 if the number of machines approaches infinity. Hence, in this case it follows that the results are optimal. We also describe the novel concept of *test-preemption*, where jobs can only be interrupted right after their test is completely executed.

Additionally, we present first results for what we call *fully-online* scheduling with testing. In this variation, all job parameters including upper bounds and testing times are unknown in the beginning and arrive sequentially one by one. The extension of List Scheduling above can be applied directly, giving a competitive ratio of 2φ. Moreover, we prove a lower bound of 2 that holds for any fixed number of machines $m \geq 2$. This bound can be increased to 2.0953 for 2 machines, and we argue that this likely holds for higher numbers of machines as well.

Results for the Sum of Completion Times. We consider this objective on one machine, where we again extend the results by Dürr et al. [35] to general testing times. We give a deterministic algorithm with competitive ratio 4, for which we also provide lower bounds that only hold for this particular algorithm. In the analysis we use novel methods of sorting the jobs by their known parameters and use cross-examination of job contributions to determine completion times. We additionally show that the methods used by Dürr et al. [35] cannot be extended to the general problem we considered. Conversely, we provide a short proof of the main deterministic result of [35] using our methods.

In the deterministic preemptive setting, we present an approach with competitive ratio $2\varphi \approx 3.2361$. For this, we customize the preemptive *Round Robin* method and apply it to our problem. Since the lower bound of Dürr et al. was not formulated for the preemptive setting, we adapt it and retain their value of 1.8546.

Finally, in the randomized setting, we devise an algorithm that is 3.3794-competitive. The randomization is applied to the testing scheme, while the ordering of the jobs is performed as in the deterministic setting. We use computer-aided computations with the software *Mathematica* [86] to calculate explicit values for the parameters of the algorithm.

For a detailed overview of all achieved contributions and a comparison to earlier results, we refer to Chapter 4. For details on our methods, please consult Chapter 5.

1.4 Publication Overview and Thesis Outline

This publication-based thesis includes two first-author papers which appeared in proceedings of peer-reviewed computer science conferences.

For every paper, we present in the appendix a short summary containing a description of the publication's content and the individual contributions of the author of this thesis. For the first paper A, the complete text including all tables and figures in the original form follows. For the second paper B, we include the final author's manuscript including

all tables and figures and a link to the published version at the publisher's web page. The bibliographic references within this thesis are [6] for paper A and [7] for paper B.

Appendix A S. Albers and A. Eckl. Explorable Uncertainty in Scheduling with Non-uniform Testing Times. In C. Kaklamanis and A. Levin, editors, *Approximation and Online Algorithms*, Lecture Notes in Computer Science, pages 127–142, Cham, 2021. Springer International Publishing. [6]

Appendix B Final author's manuscript of: S. Albers and A. Eckl. Scheduling with Testing on Multiple Identical Parallel Machines. In A. Lubiw and M. Salavatipour, editors, *Algorithms and Data Structures*, Lecture Notes in Computer Science, pages 29–42, Cham, 2021. Springer International Publishing. [7]
Published version: https://doi.org/10.1007/978-3-030-83508-8_3

For both papers, we have published *full versions* containing all detailed proofs and concepts omitted from the original publications. The full versions were originally uploaded to the free distribution service and open-access archive arxiv.org. For bibliographic information on the full versions, we refer to reference [5] for paper A and to reference [8] for paper B.

This thesis is primarily structured into six chapters: In Chapter 1 we introduced the problem setting and gave an introductory example as well as several applications. In Chapter 2, we give general definitions and describe our setting from a rigorous mathematical standpoint. Chapter 3 is dedicated to an in-depth and comprehensive review of material from the literature. We summarize all results for scheduling with testing that are known to date in Chapter 4 and place particular focus on our own results. In Chapter 5 we summarize the most important proof concepts and methods we used. Finally, we conclude with some comments, open questions, and future research possibilities in Chapter 6. In the Appendix, we include the two papers which this thesis is based upon.

2 Problem Statement

2.1 General Notions and Definitions

In this chapter we provide the basic theoretical groundwork for our problem and rigorously introduce all necessary notations, definitions, and concepts used in this thesis and the accompanying publications.

The goal of any scheduling problem is to find a *schedule* which assigns the given jobs to the machines within some given *time horizon* such that certain constraints are fulfilled. Whenever a schedule assigns some job to a machine, we also say that the job is *executed* or simply *run* on that machine.

In the scheduling with testing setting, we consider a continuous time horizon $[0, \infty)$, where jobs are assigned to machines for continuous intervals. The most important constraint that has to be fulfilled is that no two assigned job intervals may overlap on a single machine. We describe more involved constraints that have to be satisfied in Section 2.2.

A single schedule is evaluated using a given *objective function*, which is almost always based on the completion times of the jobs. For a given job, the *completion time* describes the moment in time when all processing requirements of the job are finished, i.e. the latest point anywhere on the time horizon that it is assigned to. In all problems we consider in this thesis, the goal is to *minimize* the value of the given objective function.

An *algorithm* for a given problem takes as input an instance with given parameters and outputs a schedule. We differentiate between *deterministic* and *randomized* algorithms. The latter is allowed to use random choices during its runtime to produce the schedule, while the former cannot use randomness as part of its computation.

In the scheduling with testing problem, some parts of the input are considered uncertain. We model this uncertainty by revealing information over the course of the execution of an algorithm. This means that at fixed times during the decisions of an algorithm further information about the input sequence is made available. All decisions of the algorithm must be made without any input information that has not yet been revealed. Such settings are referred to as *online*, while settings with full information about the input at all times are also referred to as *offline*. We evaluate online algorithms using the standard notion of *competitive analysis*, which was first introduced by Sleator and Tarjan [82] in 1985. It compares the algorithmic value against an optimal (offline) schedule.

Definition 2.1 (Competitive Ratio)**.** *Let $c \geq 1$, and let an online minimization scheduling problem and a deterministic algorithm ALG for the same problem be given. Let* ALG_I *denote the objective function value associated with the schedule produced by the*

algorithm on a given input sequence I and let OPT_I *denote the value of an optimal offline schedule with minimum objective function value on the same input.*

The algorithm ALG is called c-competitive *or is said to have* competitive ratio c *for the given problem if for all possible input sequences I it holds that*

$$\mathrm{ALG}_I \leq c \cdot \mathrm{OPT}_I.$$

Commonly, we drop the index I in the above notations when the input is clear by context or the statement holds for all inputs. We then simply write ALG for the algorithmic value and OPT for the optimal value, slightly abusing this notation since it also refers to the algorithm and the optimum itself.

Note that we do not explicitly demand polynomial runtime of the algorithm in the above definition. However, in many cases competitive online algorithms still fulfill this condition even though it is not required.

We also note that in some definitions of competitiveness an additive term of size $o(\mathrm{OPT}_I)$ is added to the right hand side of the inequality, leading to a slightly more general definition that is not as susceptible to difficult small instances. The above definition without this additive term is then sometimes referred to as *strictly c-competitive* [23, Chapter 1.1.2]. The two variations can in certain cases make an actual difference in the competitive ratio, see for example [16]. All proofs of competitive ratios provided in this thesis and its accompanying papers hold under the stricter definition.

For randomized algorithms, the objective value ALG_I of a schedule produced on an input sequence I is a random variable based on the algorithm's random choices. To evaluate an algorithm, we only consider settings where the input sequence is constructed in advance without any knowledge of the outcome of these random choices. In the literature, this is usually referred to as an *oblivious adversary* who plays in a two-player game against the algorithm. For more on this and other kinds of adversaries, see for example [21, 23]. We define the competitive ratio for randomized algorithms as follows using fixed input sequences:

Definition 2.2 (Competitive Ratio for Randomized Algorithms)**.** *Let $c \geq 1$, and let an online minimization scheduling problem and a randomized algorithm ALG for the same problem be given. Let* ALG_I *denote the random objective function value associated with the schedule produced by the algorithm on a given fixed input sequence I and let* OPT_I *denote the value of an optimal offline schedule with minimum objective function value on the same input.*

The algorithm ALG is called c-competitive *or is said to have* competitive ratio c *for the given problem if for all possible fixed input sequences I it holds that*

$$\mathbb{E}[\mathrm{ALG}_I] \leq c \cdot \mathrm{OPT}_I.$$

Here, $\mathbb{E}[\mathrm{ALG}_I]$ *refers to the expected value of the random variable* ALG_I*.*

Our goal is to provide online algorithms such that their competitive ratios are as small as possible, the best value of 1 corresponding to an algorithm providing an optimal

schedule for any input. Of course, achieving this value is not always possible. For competitive algorithms, impossibility results are proven based on the notion of *lower bounds*:

Definition 2.3. *Let an online minimization scheduling problem be given. Any value $l \geq 1$ such that for all c-competitive online algorithms for the problem it holds that $c \geq l$ is called a* lower bound *for this problem.*

It follows directly that any online algorithm for a problem with a lower bound of l cannot have a competitive ratio smaller than l. The ultimate goal in competitive analysis is usually to close the gap between the lower bound and the best known competitive ratio of an algorithm. Hence, the current best known ratio is also sometimes referred to as an *upper bound* to the problem. If an l-competitive algorithm is known for a problem with lower bound l, we refer to the algorithm and the lower bound as *tight*.

To prove lower bounds for deterministic algorithms, it is very common to describe an example input where any possible online algorithm returns a schedule whose value is at least l times larger than the optimum. We note that for the more general definition of competitiveness with an additive term technically a sequence of examples with increasing size has to be provided to prove a valid lower bound. It is easy to check that all given lower bound examples in this thesis and the appended papers can be extended to such an increasing sequence of inputs, hence all provided lower bounds hold even for the more general version of competitiveness.

For randomized algorithms, a lower bound can theoretically be achieved in the same fashion. However, when describing the counterexample one has to ensure not only that any online algorithm has an expected objective value at least l times larger than the optimum, but this also has to be true for any probability distribution of its random choices. Since this can sometimes be difficult to achieve, lower bounds for randomized algorithms are often proven using *Yao's principle* [87]. For more details, see for example [23, Chapter 8.3].

For a complete and in-depth overview of online algorithms and competitive analysis, we refer the reader to the book by Borodin and El-Yaniv [23]. In the next section we describe the parameters and notations of the scheduling with testing setting in elaborate detail.

2.2 Description of Scheduling with Testing

An instance of the scheduling with testing problem consists of n jobs that have to be assigned to m machines, where $m, n \in \mathbb{N}$. We usually use the index j for a job and the index i for a machine. Within some proofs, index k is also used to refer to a job.

All jobs j have a *processing time* $p_j \in \mathbb{R}_0^+$, which is unknown to the algorithm at the beginning of its execution. Every job j also has an *upper bound* $u_j \in \mathbb{R}_0^+$ and a *testing time* $t_j \in \mathbb{R}_0^+$. Unlike the processing times, the upper bounds and testing times are known to the algorithm from the start. In particular, it follows that the algorithm knows the number of jobs n. As indicated by the name, the upper bound of a job

provides a maximum size for the corresponding processing time, or in other words it always holds that

$$0 \leq p_j \leq u_j, \quad \forall j \in [n]. \tag{1}$$

Any job j must be assigned to the machines in one of two possible configurations: it is either run *untested* or *tested*. In the untested configuration, the total time the job has to be executed is u_j. If the job is tested, it has to be executed for a total time of $t_j + p_j$. Here, the uncertainty of the parameter p_j comes into play to make our problem an *online* one: Only as soon as the testing time t_j is fully completed on some machine, the value of p_j is revealed to the algorithm. Depending on the specific setting, the algorithm can then reevaluate its decision for the assignment of the processing time that it just received. When the testing time is already larger than the upper bound, i.e. $t_j > u_j$, we say the job is *trivial*. It is clear that trivial jobs are always run untested. Furthermore, an instance is called *uniform*, if all testing times t_j are equal to 1.

Once an algorithm ALG has assigned all jobs, we can determine the *objective value* of the resulting schedule. As mentioned in the previous section, we define the *completion time* C_j^{ALG} of a job j as the maximum time on the time horizon where an assignment of j to some machine is made. Usually, the algorithm and the resulting schedule are clear by context, and we simply write C_j. In rare cases when we refer to the completion time of j under the schedule produced by the optimum, we use C_j^{OPT} instead.

Definition 2.4 (Objective Functions)**.** *Let an instance of scheduling with testing with n jobs and a corresponding schedule be given. We define the* makespan *objective function as*

$$\max_{j \in [n]} C_j,$$

and the sum of completion times *objective function as*

$$\sum_{j=1}^{n} C_j.$$

It can be seen directly that in case there is only a single job, i.e. $n = 1$, the two objectives are actually equivalent. In almost all other scenarios, it will turn out that optimizing the sum of completion times is harder than the makespan.

Assignment of Jobs to Machines. In the next paragraphs, we describe the specific rules of how a job can be assigned to the machines. This depends not only on its configuration, but also whether we allow jobs to be *interrupted*.

In all settings, the following conditions must hold: Every machine can only schedule a single job at once, and a single job cannot be assigned to more than one machine at the same time. The configuration of a job has to be decided before any part of the job is assigned. Additionally, in case a job is run in the tested configuration, any section of the processing of the job can only happen after any section of the test. This corresponds to the intuitive idea that the test - which reveals the processing time - has to be fully completed in order to impart the information. Finally, we differentiate between three settings pertaining to the interruption of jobs:

In the *non-preemptive* setting, jobs are not allowed to be interrupted and have to be assigned to a single uninterrupted interval on one machine. The length of the assignment is u_j in the untested configuration and $t_j + p_j$ in the tested configuration.

In the *test-preemptive* setting, only tested jobs can be interrupted, and only right after their test is completed. Hence, in the untested configuration, a job has to be assigned to an uninterrupted interval of length u_j on a single machine. In the tested configuration, the test has to be assigned to one machine for an uninterrupted interval of length t_j. The following processing time can then be assigned to some other machine for an uninterrupted interval of length p_j.

Lastly, in the *preemptive* setting, jobs can be interrupted at any time independent of their configuration and can be assigned to an unbounded number of different intervals on all machines for a total time depending on the job configuration: the total assignment length is u_j in the untested configuration and $t_j + p_j$ in the tested configuration.

It is easy to see that the difficulty of these settings decreases in the order of non-preemptive, test-preemptive, and preemptive, since the opportunities of interrupting jobs are increasing. In particular, an algorithm for a more difficult setting can always be applied to an easier setting by just ignoring the additional possibility of interrupting jobs.

Even though jobs are always assigned to fixed intervals according to the rules established in the previous paragraphs, we might sometimes make shortcuts when describing this assignment for a given objective function. For example, it is clear that it makes no sense for an algorithm to leave empty space somewhere on a machine only to resume executing jobs later on that same machine. Instead, it could just move all jobs forward until the gap is filled. Therefore, we assume that any algorithm we describe *always* fulfills this very simple condition that it leaves no gaps in the schedule on any machine. Using this, we can simplify the description of how an algorithm assigns jobs in certain situations:

For the makespan objective, we only need to describe which machine a job is assigned to, and then assume that the jobs are run on this machine in some order without any gaps. Since the order on the machines does not influence the maximum completion time over all jobs, this is a sufficient description of a unique solution for the makespan.

For the sum of completion times, the order of the jobs on a machine does indeed matter, and therefore we need to specify it when describing a solution. However, the exact start and end times of the assigned intervals can again be omitted, because we may assume all jobs are scheduled one after another in the specified order without leaving any gaps.

Fully-Online Variant of Scheduling with Testing. We now define a special case of our problem, the so-called *fully-online* variant. In this version, the jobs are not known to the scheduler in the beginning, and are only revealed one by one. Therefore, an algorithm does not know the total number of jobs, and only learns the values of upper bound and testing time at the moment when a new job arrives. Only if the job is tested, its processing time is revealed. After completely and irrevocably assigning a job to the

machines, the next job is presented to the algorithm. All other aspects of this setting are equivalent to the normal version of scheduling with testing.

Job Variables. Before we conclude this section, we define some useful variables based on the already described parameters. These variables will appear frequently in our algorithms and proofs.

The first such variable is the *parameter ratio* r_j between the upper bound and the testing time. Let a job j be given. Then

$$r_j := \frac{u_j}{t_j}. \tag{2}$$

If $t_j = 0$, we define $r_j = "+\infty"$. It is clear that for non-trivial jobs it holds that $r_j \geq 1$. We will use the parameter ratio quite often to define testing strategies in our algorithms. In connection to the ratio, the mathematical constant of the *golden ratio* will appear frequently. We define $\varphi \in \mathbb{R}^+$ as the unique positive real number fulfilling the equation $\varphi = 1 + 1/\varphi$. It holds

$$\varphi = \frac{1+\sqrt{5}}{2} \approx 1.6180.$$

Next, we describe the three *running time* variables of jobs:

Definition 2.5 (Job Running Times)**.** *Let an instance of scheduling with testing and an online algorithm ALG be given. Additionally, let j be a given job.*

The (optimal) offline running time ρ_j *of job j is defined as*

$$\rho_j := \min(t_j + p_j, u_j).$$

The algorithmic running time p_j^A *of job j is defined as*

$$p_j^A := \begin{cases} t_j + p_j & \text{if } j \text{ is tested by ALG,} \\ u_j & \text{if } j \text{ is not tested by ALG.} \end{cases}$$

The minimal running time τ_j *of job j is defined as*

$$\tau_j := \min(t_j, u_j).$$

The job running times are directly related to the lengths of jobs under different schedules: A schedule with optimal objective value, against which we compare the algorithmic solution, is constructed with full knowledge about the parameters of the problem, in particular the processing times p_j. Therefore, we can directly determine the best configuration of a single job j: If $p_j + t_j < u_j$, then test the job, otherwise run it untested. Hence, the optimal offline running time ρ_j corresponds to the time the optimum needs to run job j.

A given online algorithm ALG, on the other hand, does *not* know the processing times p_j. It has to non-trivially decide whether a given job j is tested or not. Therefore,

the algorithmic running time p_j^A, i.e. the time needed for ALG to execute job j on a machine, is dependent on the used configuration. Usually, the associated algorithm for p_j^A is clear by context, and we only specifically mention it in ambiguous cases.

Finally, the minimal running time τ_j is a lower bound for the minimum time *any* schedule has to spend on running job j. The minimal running time is based only on known parameters of the problem, which means it can be computed for all jobs by any algorithm. It will be used frequently for sorting or categorizing the jobs when the processing times are not yet known.

One main aspect of any competitive algorithm will be how well it can bound the algorithmic running times against the optimal offline running times. When there is more than one job or multiple machines, a second goal will be to sort and assign the jobs in a suitable way. We will see simple algorithms that tackle these challenges separately, as well as more sophisticated algorithms where the job assignment is influenced by the testing decision and vice versa.

3 Literature Review

3.1 Scheduling with Testing

We begin the literature section with a review of the scheduling with testing model itself. In 2018, Dürr et al. [35] published the paper 'Scheduling with Explorable Uncertainty' where they first introduced the new model motivated by real life settings where some preliminary test can be applied to jobs before they are executed. Their results focused on scheduling on a single machine where all testing times are equal to 1, a setting we refer to as *uniform testing*. They later published a journal version [34] of their paper.

The main results of Dürr et al. for the objective of minimizing the sum of completion times are: The algorithm Threshold which is 2-competitive in the deterministic setting, and the algorithm Random which has an improved ratio of 1.7453 in the randomized setting. Additionally, they presented algorithms with improved ratios for the deterministic setting if all upper bound parameters have the same value. They completed their results with a deterministic lower bound of 1.8546, which holds even in the case when all upper bound parameters are equal and the processing times are either 0 or equal to the upper bound. Finally, they also provided a randomized lower bound with a value of 1.6257.

For the makespan objective, Dürr et al. provided a ratio of $\varphi \approx 1.6180$ in the deterministic setting as well as a ratio of 4/3 in the randomized setting. For both settings, they also provided tight counterexamples.

Dürr et al. acknowledged that "In some applications, it may be appropriate to allow the time for testing a job to be different for different jobs [...]" [35] and left the investigation of this general case up to future research. In our paper [6] from 2021 we studied the case with non-uniform testing times $t_j \geq 0$ on one machine. For the sum of completion times, we presented the deterministic 4-competitive algorithm (α, β)-SORT as well as the randomized 3.3784-competitive algorithm Randomized-SORT. For both cases, the lower bounds from the uniform case apply. Contrary to Dürr et al., we also considered the preemptive setting and proved a 3.2361-competitive algorithm called Golden Round Robin. We additionally extended the uniform deterministic lower bound to the preemptive setting. Finally, we extended the makespan results of the uniform case to general testing, achieving the same ratios as Dürr et al. [35].

More recently, we generalized the problem further and considered multiple machines in our paper [7] published later in 2021. We focused on the makespan objective, a setting which is quite similar to the classic online makespan minimization problem. The main results from this paper are a 3.1016-competitive algorithm in the non-preemptive setting which can improved to a 3-competitive algorithm in the case of uniform tests.

We compare these results with a simple extension of List Scheduling, which is 3.2361-competitive, and a lower bound that approaches 2 if the number of machines becomes large. We additionally considered settings where jobs can be interrupted, and achieved a 2-competitive algorithm called Two Phases for the test-preemptive setting, as well as a lower bound that approaches 2 for the ordinary preemptive case.

The achievements from the papers [6] and [7] are the main focus of this thesis. We refer to Chapter 4 for an in-depth overview of all results, and to Chapter 5 for a high level introduction to the methods used.

3.2 Explorable Uncertainty

Explorable uncertainty is a specific model of handling uncertainty, which was introduced in a seminal paper by Kahan [59] in 1991. It is sometimes also referred to as *queryable uncertainty* and describes settings where some query can be used to receive more information about the uncertain input. The algorithm can *explore* this yet unavailable information by investing resources into the query.

We first consider query-competitive algorithms, i.e. algorithms with the objective of minimizing the number of queries. Kahan's work [59] studied uncertain values obtained from given closed intervals, and derived approximation guarantees for the number of queries needed to obtain the maximum, median or minimum distance of the values. Following his work, other publications studying selection problems include Khanna and Tan [65], who computed the average and the k-smallest value, Feder et al. [45], who also studied the k-smallest value, and Charikar et al. [26], who evaluated boolean AND/OR trees and search functions for sorted arrays. Very recently, Erlebach et al. [39] considered selection problems with rounds of parallel queries.

In 2005, Bruce et al. [24] considered geometric tasks, more specifically finding all maximal points or all points lying on the boundary of the convex hull of an uncertain two-dimensional input set. Additionally, Bruce et al. introduced the notion of *witness sets*: In order to obtain a solution, at least one value from every witness set has to be queried. This concept was later stated in more general form by Erlebach et al. [42].

Olston and Widom [71] developed a system for computing query answers in distributed applications, where the solution is allowed to be inaccurate up to some specified precision constraint. Sorting in the query-competitive setting was examined recently by Halldórsson and de Lima [55]. Arantes et al. [14] studied scheduling with uncertain error occurrences, where the number of queries used to detect time slots with errors is minimized.

Quite a few combinatorial problems have also been studied: One such problem is minimum spanning trees, which was studied by Erlebach et al. [42], Megow et al. [68], as well as Gupta et al. [54]. Feder et al. [44] considered shortest paths, while Goerigk et al. [50] investigated the knapsack problem. Erlebach et al. [41] and Merino and Soto [70] researched methods for finding minimum-weight bases of matroids and more general set collections.

A very recent line of research is concerned with online algorithms using machine-

learned predictions. We refer to the paper by Erlebach et al. [40], which combined this new area with explorable uncertainty and references therein.

The survey from 2015 by Erlebach and Hoffmann [38] contains an overview of research on query-competitive algorithms.

A related but slightly different setting are tasks where the query cost is included in the objective function of the corresponding optimization problem. Most importantly, the scheduling with testing problem that was considered in [6, 7, 35] and is the main topic of this thesis falls into this category: To receive more information about the processing time of a job, the corresponding test has to be executed on some machine of the instance and directly counts towards the given objective based on the completion time of the jobs. A framework that is very close to this setting is the processing time oracle model introduced recently by Dufossé et al. [33].

Another example for such settings is the so-called Pandora's Box problem (Weitzman [85]). Herein, a set of random variables are explored with the goal of maximizing the highest revealed value, where the costs of revealing a value are subtracted from the collected reward.

In 2018, Singla [81] introduced his 'price of information' model, where he examined combinatorial problems with stochastic uncertainty. Information is received in exchange for paying a probing price, which is again subtracted from the given objective. More recently, Gupta et al. [53] generalized this model to combinatorial optimization problems with multiple stages of stochastic uncertainty.

Finally, there are many publications in the so-called *multi-armed bandit* framework, which studies the trade-off between exploration and exploitation, mostly under stochastic uncertainty models. For an overview, we refer to the monograph by Bubeck and Cesa-Bianchi [25] and the book by Gittins et al. [49]. We highlight a paper by Levi et al. [67], who studied a model where testing jobs with stochastic parameters provides additional information to the scheduler. They presented structural insights and (near-)optimal results.

3.3 Scheduling on a Single Machine

Scheduling in the single machine case is of interest for us, since we consider scheduling with testing on one machine extensively in our paper [6], both for the makespan and the sum of completion times objective. In this section, we briefly give an overview of previous results for these two objectives on one machine.

In its classic form without any additional uncertainty, the makespan objective can be solved optimally by any algorithm that does not leave any gaps in the schedule. This holds true even if the jobs arrive one by one in an online setting.

For the sum of completion times, the picture is more varied. We restrict our attention to results without job weights and first look at non-preemptive algorithms. In the offline variant, the *Shortest Processing Time first* (SPT) algorithm is optimal, as has been proven by Smith [83] as early as 1956. Apart from that, there are several results with *release times*, where jobs may only be assigned to the machine after a given point on

the time horizon. The offline problem with release times is strongly NP-hard (Lenstra et al. [66]) and can be solved with a polynomial-time approximation scheme (Afrati et al. [1]) up to an arbitrarily small relative error in the objective function. In the online version, jobs are only revealed to the algorithm at their release time. For this setting, a lower bound of $2 - \varepsilon$ was provided by Hoogeveen and Vestjens [57]. A tight upper bound of 2 was given in the same paper and independently by Phillips et al. [72].

The *preemptive* setting is easier than its non-preemptive counterpart. The *Shortest Remaining Processing Time first* (SRPT) algorithm is optimal in the presence of release times and in the online version (Schrage [78]). One algorithm of note in the preemptive online one machine setting is the so-called *Round Robin* algorithm. It continually cycles through all available jobs, running every job for a very small, fixed amount of time on the machine before moving on to the next job. The competitive ratio of this algorithm in the preemptive setting is 2 [73]. In our paper [6], we adapt this algorithm for scheduling with testing in the preemptive case on one machine.

3.4 Online Makespan Minimization

In our second paper [7], we consider scheduling with testing on multiple machines with the objective of minimizing the makespan. In its offline form, makespan minimization on m machines has been proven to be NP-hard [48]. There exists a well-known polynomial-time approximation scheme by Hochbaum and Shmoys [56].

The online variant of makespan minimization in its basic form has been investigated in great detail in the last decades. Already in 1966, Graham [51] published his well-known List Scheduling algorithm, which is $(2 - 1/m)$-competitive in the deterministic setting.

The systematic initiation of competitive analysis following the 1985 paper by Sleator and Tarjan [82] lead to the problem being studied extensively and Graham's deterministic upper bound was improved multiple times: Galambos and Woeginger [47] proved a slightly smaller ratio of $2 - 1/m - \varepsilon_m$ where $\varepsilon_m \to 0$ for $m \to \infty$. The first result beating 2 even for large values of m came from Bartal et al. [17] with a ratio of 1.985. This was improved to 1.945 by Karger et al. [61] and then to 1.923 by Albers [2], before Fleischer and Wahl [46] presented the best currently known result of 1.9201 in 2000.

The lower bound was likewise the focus of many publications. Faigle et al. [43] proved a bound of 1.707, which was improved to 1.837 by Bartal et al. [18] and then to 1.852 by Albers [2]. Rudin [76] gave the current best lower bound of 1.88 in his PhD thesis.

For the randomized version of online makespan minimization, an upper bound of 1.916 was shown by Albers [3]. For the lower bound, a value of $e/(e-1) \approx 1.582$ was proven independently by Chen et al. [27] and Sgall [80]. In the deterministic preemptive setting, there exists a tight bound of $e/(e-1)$ for large m by Chen et al. [28].

There are numerous variations of online makespan minimization that have been introduced in more recent years, like for example in *semi-online scheduling* or *resource augmentation*. We summarize a selection of publications for these two variants in separate sections below.

3.5 Semi-online Scheduling

In semi-online settings, the algorithm has access to some additional piece of information of the instance, e.g. the sum of all processing times, the job order, or the value of the optimum. Scheduling in the semi-online setting is of particular interest for our problem, since we may consider our upper bounds and testing times as the extra pieces of information an algorithm receives about the unknown processing times of the jobs, making our problem a semi-online one from a certain viewpoint.

We focus our attention on deterministic settings with makespan minimization on multiple identical machines. Graham [52] considered makespan minimization with jobs sorted by processing time as early as 1969 and presented the well-known *Longest Processing Time first* (LPT) algorithm. More recently, Cheng et al. [29] also considered this setting and improved Graham's result. Additional information about the sum of all processing times was first considered in Kellerer et al. [64], recent optimal results for m machines are found in Albers and Hellwig [9] and Kellerer et al. [63]. The case when the additional piece of information corresponds to the value of the optimum was studied by Azar and Regev [15]. In this variant, the problem is also known as *bin stretching* [15]. Böhm et al. [22] presented the best known algorithm for the problem.

A recent survey by Epstein [37] gives a detailed overview of publications regarding deterministic semi-online scheduling.

3.6 Resource Augmentation

In resource augmentation settings, the algorithm receives access to some extra resources, for example machines with higher speed or a reordering buffer. Compared to semi-online scheduling, the algorithm now has no extra input information, but can instead alter the machines, the schedule, or other characteristics of the instance.

We again provide some examples of papers that examine resource augmentation for makespan minimization on identical machines: With a so-called *reordering buffer*, an algorithm can temporarily store jobs before they are assigned. Kellerer et al. [64] and Zhang [88] initiated this setting in 1997. The most important result on reordering buffers was presented by Englert et al. [36]. Kellerer et al. [64] also proposed the idea of *parallel schedules* where an algorithm can compute several schedules in parallel and then use the best solution it obtained. Albers and Hellwig [11] provided more insights for this setting. Parallel schedules are closely related to settings with *advice*, where the algorithms can access a number of bits telling it how to behave. Dohrau [32] studied this framework in the online makespan minimization setting. For settings where machines with higher speed are available to the algorithm, we mention the papers by Kalyanasundaram and Pruhs [60] and Anand et al. [13]. Finally, an algorithm might be allowed to *migrate* jobs, that is to remove and reschedule already assigned jobs, as for example in Sanders et al. [77] or Albers and Hellwig [10].

More details on resource augmentation can also be found in the surveys by Epstein [37] and Albers [4].

3.7 Other Notable Publications in Online and Stochastic Optimization

In this section, we provide some more publications for the online and stochastic optimization setting and highlight a few papers that are similar to our problem.

A version of online scheduling that is closely related to scheduling with testing is *scheduling with rejections*. Here, a job may either be scheduled on a machine or alternatively be rejected at the cost of a predetermined penalty. The balance between the fixed penalty and an increase in the objective is reminiscent of the testing decision an algorithm has to make in the scheduling with testing setting. Bartal et al. [19] considered scheduling with rejections using deterministic non-preemptive algorithms with the goal of minimizing the makespan, while Seiden [79] considered the corresponding preemptive setting. Just as in our problem, many competitive ratios depend on the golden ratio, again illustrating the balancing decisions based on job parameters an algorithm has to address in both scenarios. However, contrary to our setting, scheduling with rejections only contains uncertainty in the form of online arrivals, no parameters are unknown or have to be explored via a test.

Another interesting setting investigated by Albers and Janke [12] in 2021 is online makespan minimization under budgeted uncertainty. In this framework, a constant number of jobs may fail and require additional processing time. For more information and previous work on budgeted uncertainty, we refer to the references in [12].

Some historical notes, a general overview, and several related settings for online scheduling are discussed in Pruhs et al. [75].

To conclude this chapter, we briefly give some references for *stochastic optimization*. The topic is related to our setting since some additional information on the unknown input is given by a known probability distribution. Early work in stochastic optimization includes Dantzig [31] and Beale [20]. We also refer to the chapters on stochastic scheduling models in Pinedo [73]. We conclude with two papers that first combined the frameworks of online and stochastic optimization in scheduling and refer to contained references: Chou et al. [30] considered the sum of completion times on a single machine, while Megow et al. [69] studied identical parallel machines.

4 Summary of Previous and New Results for Scheduling with Testing

We now summarize all currently known results for scheduling with testing. For the makespan objective, we present results for a single machine as well as multiple identical machines. For the sum of completion times objective, we give results for the case of a single machine.

4.1 Makespan

We start by presenting the currently known results for the scheduling with testing problem for the makespan objective on a single machine. Dürr et al. [35] initiated the line of research into scheduling with testing in 2018 by considering this setting with uniform testing times $t_j \equiv 1$. They provided a φ-competitive deterministic algorithm based on a simple testing scheme that evaluates the known parameters of a single job. Here, $\varphi \approx 1.6180$ is the golden ratio. They complemented this result with a tight lower bound of the same value. For the setting where an algorithm can make randomized decisions, Dürr et al. provided a 4/3-competitive algorithm and again a tight lower bound. In our paper [6], we extended their results to general testing times $t_j \geq 0$ and achieved the same ratios. It follows that our results are tight as well, since the lower bounds by Dürr et al. can be directly extended to the more general case. All results for the makespan on a single machine are summarized in Table 2. We take a closer look at the methods used for the one machine case in Section 5.2.1.

Table 2: Results for the makespan objective on a single machine.

Algorithm	**General Tests**	**Uniform Tests**	**Lower Bound**
deterministic	$\varphi \approx 1.6180$ [6]	φ [35]	φ [35]
randomized	$\frac{4}{3}$ [6]	$\frac{4}{3}$ [35]	$\frac{4}{3}$ [35]

We note that for the makespan objective on one machine, the distinction between non-preemptive and (test-)preemptive settings is unnecessary since they are all equivalent. This follows directly from the fact that the order of executions on a single machine does not influence the objective value of the makespan as long as there are no gaps between the jobs. For more than one machine, however, this changes because jobs can then be assigned to a different machine after being interrupted.

In our paper [7] and its full version [8], we presented deterministic algorithms for the makespan objective for any number of machines $m \in \mathbb{N}$ and examined non-preemptive, test-preemptive, and preemptive settings. We did not consider any randomized algorithms for multiple machines.

For the non-preemptive setting, we presented a sophisticated method called SBS algorithm, whose competitive ratio $c(m)$ increases with the number of machines m and reaches a maximum value of 3.1016 for $m \to \infty$. For uniform testing times, the same algorithm can be adapted to be $c_1(m)$-competitive, where the function $c_1(m)$ is also increasing in m and has a maximum value of 3 if m approaches infinity. Additionally, we compared our results to a method based on the List Scheduling algorithm. This so-called Extended List Scheduling algorithm is $\varphi(2 - 1/m)$-competitive, a value that approaches 3.2361 for large m. The analysis of the algorithm is tight, i.e. there exist specific examples where the ratio is not better than $\varphi(2 - 1/m)$. To complement our results, we provided a lower bound of $\max(\varphi, 2 - 1/m)$. As an additional minor result, we introduced an algorithm for the case when there are at most m non-trivial jobs. This algorithm has a competitive ratio of $\varphi(4/3 - 1/(3m))$, a value which approaches 2.1574 if m becomes large. Note that the lower bound above does *not* apply to this specialized case since it requires a higher number of non-trivial jobs.

In Table 3, we give an overview of the competitive ratios of the Extended List Scheduling algorithm, the SBS algorithm in both general and uniform form, and the lower bound and how they vary for different values of m. For completeness, we also include the ratios of the special algorithm for instances with few non-trivial jobs.

Table 3: Deterministic non-preemptive results from [7] and [8] for fixed values of m.

	$m = 1$	$m = 2$	$m = 3$	$m = 4$	$m = 5$	$m = 10$	$m = 100$	$m \to \infty$
Extended LS	1.6180	2.4271	2.6967	2.8316	2.9125	3.0743	3.2199	3.2361
SBS	1.6180	2.3806	2.6235	2.7439	2.8158	2.9591	3.0874	3.1016
Uniform-SBS	1.6180	2.3112	2.5412	2.6560	2.7248	2.8625	2.9862	3
Lower Bound	1.6180	1.6180	1.6667	1.75	1.8	1.9	1.99	2
Few Non-trivial Jobs	1.6180	1.8877	1.9776	2.0225	2.0495	2.1034	2.1520	2.1574

As we can see, all algorithms from [7] and [8] achieve the same tight ratio of $\varphi \approx 1.6180$ for $m = 1$ as was already proven in [35]. Additionally, it is apparent that the simple Extended List Scheduling method is outperformed by the SBS algorithm for all values of $m \geq 2$. In turn, the uniform version of SBS improves the results even further. Finally, it is not surprising that the algorithm for the case with at most m non-trivial jobs provides the best ratios. In Figure 2, we additionally present the graphs of the functions $\varphi(2 - 1/m)$, $c(m)$, $c_1(m)$, $\varphi(4/3 - 1/(3m))$, and $\max(\varphi, 2 - 1/m)$ in dependence of m.

For the non-preemptive setting, on the other hand, we introduced almost tight results: The Two Phases algorithm has a competitive ratio of 2 for any value of m, both for general and uniform testing times. Additionally, this algorithm works even in the more

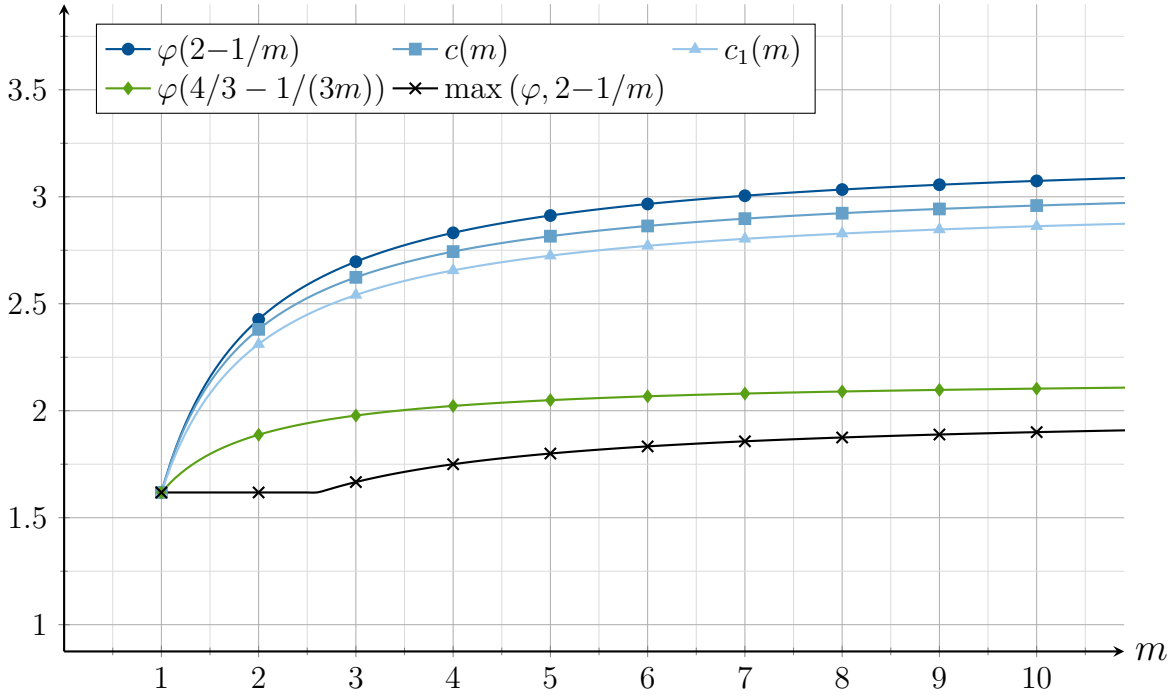


Figure 2: Function plots of the competitive ratios of Extended List Scheduling, SBS, Uniform-SBS, and the algorithm for instances with few non-trivial jobs as well as the function plot of the lower bound.

restrictive test-preemptive setting. We also proved a corresponding lower bound of $\max(\varphi, 2 - 2/m + 1/m^2)$ for the ordinary preemptive setting which can be directly extended to the test-preemptive setting. Hence, this bound is tight if the number of machines approaches infinity.

The best currently known results for the makespan objective on multiple machines are summarized in Table 4. Please consult Sections 5.2.2 and 5.2.3 for more details on the methods we applied in the proofs.

Table 4: Deterministic results from [7] for the makespan on m machines.

Setting	**General Tests**	**Uniform Tests**	**Lower Bound**
non-preemptive	$c(m) \xrightarrow[m\to\infty]{} 3.1016$	$c_1(m) \xrightarrow[m\to\infty]{} 3$	$\max\left(\varphi, 2 - \frac{1}{m}\right)$
(test-)preemptive	2	2	$\max\left(\varphi, 2 - \frac{2}{m} + \frac{1}{m^2}\right)$

Finally, we briefly examined the *fully-online* variation where all jobs are revealed to the algorithm one by one in the full version [8]. We only considered the test-preemptive setting. Extended List Scheduling can be applied directly, giving a competitive ratio of 2φ in the worst case. We also provided a lower bound with a value of 2 that holds for all $m \geq 2$. If $m = 2$, this bound can be increased to 2.0953. For more details, we refer to Section 5.2.4.

4.2 Sum of Completion Times

In this section, we summarize the results on a single machine for the objective of minimizing the sum of completion times. We first focus on the uniform testing case, for which Dürr et al. [35] provided the first results when they introduced the problem in 2018. We present these results in Table 5. They described a deterministic 2-competitive algorithm called Threshold as well as a deterministic lower bound of 1.8546. In the randomized setting, they provided an algorithm Random with competitive ratio of 1.7453 and a lower bound of 1.6257. Additionally, they provided several special case algorithms for uniform job bounds that are closer to the lower bound of 1.8546.

Table 5: Results from [35] for the sum of completion times with uniform tests.

Algorithm	Setting	Competitive Ratio	Lower Bound
deterministic	test-preemptive	2	1.8546
randomized	test-preemptive	1.7453	1.6257

It should be noted that all results from [35] are presented for what we call the test-preemptive setting, even though Dürr et al. do not make this distinction. In other words, in their setting jobs are allowed to be interrupted after the test is completed, a concept which they call *deferring* or *delaying* a job. For the strictly non-preemptive setting, no deterministic algorithm with finite competitive ratio can exist, even on one machine with uniform testing. This result is new in this thesis and was not presented before. We provide the proof, which is based on a counterexample, and the relevant mathematical details in Section 5.3.3.

In the non-uniform testing case, we have several findings to report that were achieved in our paper [6] and its full version [5]. As our main result, we provided a deterministic test-preemptive 4-competitive algorithm called $(1, 1)$-SORT. This result is complemented by the lower bound from [35] for the uniform case, which can be extended to general tests directly. We also proved that the algorithm itself cannot be better than 3-competitive. Even if some parameters are modified in the algorithm, it can still not be better than 2-competitive.

Additionally, we considered the deterministic preemptive setting where we achieved a 2φ-competitive algorithm. The analysis of the algorithm is tight. We also non-trivially adopted the corresponding result by Dürr et al. to achieve a lower bound of 1.8546 for this setting.

For randomized algorithms, we only considered the test-preemptive setting, where we provided a 3.3794-competitive approach. Here, the lower bound by Dürr et al. [35] can again be applied directly.

Consult Table 6 for an overview of the achieved ratios for the sum of completion times with general tests. Please refer to Sections 5.3.1, 5.3.2, and 5.3.4 for details on the methods used in the proofs.

Table 6: Results for the sum of completion times with general tests.

Algorithm	Setting	Competitive Ratio	Lower Bound
deterministic	non-preemptive	no finite ratio possible	
deterministic	test-preemptive	4 [6]	1.8546 [35]
deterministic	preemptive	$2\varphi \approx 3.2361$ [6]	1.8546 [6]
randomized	test-preemptive	3.3794 [6]	1.6257 [35]

4.3 Comparing Results for Both Objectives

To conclude this section, we compare some of the results presented above. We start by comparing the two different objective functions that we considered for the one machine case. As we have mentioned previously in Section 2.2, the makespan and the sum of completion times are equivalent if there is only a single job.

For more than one job in the makespan case, there is a tight deterministic algorithmic solution with ratio $\varphi \approx 1.6180$, as can be seen in Table 2. On the other hand, for the deterministic (test-)preemptive sum of completion times there is a lower bound equal to 1.8546 (c.f. Tables 5, 6). Hence, in the deterministic setting, minimizing the sum of completion times is strictly harder than minimizing the makespan on a single machine with more than one job.

This holds for randomized algorithms as well: The tight algorithmic solution for the makespan has a ratio of 4/3 (c.f. Table 2), while the lower bound for the sum of completion times is 1.6257 (c.f. Tables 5, 6). Again, minimizing the sum of completion times is strictly harder than minimizing the makespan if there is more than one job on a single machine.

In addition, we compare the different settings of how jobs can be assigned to the machines. It is clear that the difficulty of scheduling with testing decreases in the following order: non-preemptive, test-preemptive, and preemptive. This can also be seen in the corresponding results in Tables 4 and 6.

Similarly, allowing an algorithm to make random choices is stronger than not permitting randomness. In the one machine case for the makespan and in the case of uniform testing times for the sum of completion times, we already know that randomization leads to the problem being strictly easier (c.f. Tables 2, 5). Comparatively, it is still unclear whether randomization leads to a strictly easier problem in the case of general tests for the sum of completion times (c.f. Table 6).

Finally, the test-preemptive fully-online variation of the problem has a lower bound that is strictly larger than the corresponding bound for the ordinary version. However, it is again unclear whether the fully-online setting itself is strictly harder or not, since there is no known competitive algorithm in the ordinary version beating this bound.

5 Methodology

In this chapter we report the most important methodological aspects of our results. For each statement in this chapter, we assume that all definitions for the scheduling with testing setting are given as described in Chapter 2. We focus on the concepts and ideas of our approaches and only give complete proofs whenever this contributes to the understanding of the associated method. Unless otherwise stated, all results in this chapter originate from the papers [6] and [7]. At times, we have slightly adjusted the wording within the cited statements to better fit into the streamlined description of the thesis. Omitted proofs can be found in the papers or in the corresponding full versions.

5.1 Important General Results

This section provides a few introductory results which are used in many of the subsequent proofs. They summarize various findings from [6] and [7] that were not given as explicit statements. The first two results will be on the values of our objective functions on one machine.

Lemma 5.1. *Let an instance of scheduling with testing on a single machine, an online algorithm ALG, and the offline optimum OPT be given. The objective values for the makespan can be written as*

$$\mathrm{ALG} = \sum_{j=1}^{n} p_j^A, \qquad \mathrm{OPT} = \sum_{j=1}^{n} \rho_j.$$

Proof. We first recall from Chapter 2 that the time an online algorithm ALG needs for a single job j was given by p_j^A, while the time the offline optimum OPT needs for j was given by ρ_j. We also assumed without loss of generality that all algorithms leave no gaps in the schedule and argued that the makespan objective value does not depend on the job order.

By Definition 2.4, the makespan of the algorithm is $\max_{j\in[n]} C_j^{\mathrm{ALG}}$. Since the algorithm leaves no gaps in the schedule, this is the same as the sum over the algorithmic running times of all jobs:

$$\mathrm{ALG} = \max_{j\in[n]} C_j^{\mathrm{ALG}} = \sum_{j=1}^{n} p_j^A$$

This can be done equivalently for the optimum, which clearly also leaves no gaps in the schedule:

$$\mathrm{OPT} = \max_{j\in[n]} C_j^{\mathrm{OPT}} = \sum_{j=1}^{n} \rho_j$$

□

Lemma 5.2. *Let an instance of scheduling with testing on a single machine and the offline optimum OPT be given. Additionally, let the order of the jobs be such that* $\rho_1 \geq \cdots \geq \rho_n$. *For the sum of completion times, the objective value of OPT can be written as*

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

Proof. Again note that the duration OPT needs for a single job j is given by ρ_j, and OPT leaves no gaps in the schedule. In contrast to the makespan, the sum of completion times objective does indeed depend on the job order. It is clear that the optimal strategy is to sort the jobs by non-decreasing optimal offline running time. Since $\rho_1 \geq \cdots \geq \rho_n$, the schedule on the machine is therefore given by $n, n-1, \ldots, 2, 1$. For a given job j, the completion time under the optimum is

$$C_j^{\mathrm{OPT}} = \sum_{i=j}^{n} \rho_i.$$

To receive the actual value of the optimum we sum up all these value according to Definition 2.4 and apply some elemental algebra:

$$\mathrm{OPT} = \sum_{j=1}^{n} C_j^{\mathrm{OPT}} = \sum_{j=1}^{n}\sum_{i=j}^{n} \rho_i = \sum_{j=1}^{n} j \cdot \rho_j$$

□

The above Lemma can also be reformulated to include the formula for the objective value of an algorithm. However, this would require re-arranging the jobs by non-increasing size of the algorithmic running times. Since an actual representation of the algorithmic value is rarely needed in our methods, we forgo the necessary re-indexing and only present the far more important optimal value.

We continue with two important, general observations on the relationships of the different *running times*. The following proposition contains a number of statements from [6] and [7]:

Proposition 5.3. *Let an instance of scheduling with testing and an online algorithm ALG be given. Then the following holds:*

$$\tau_j \leq \rho_j \leq p_j^A, \qquad \forall j \in [n] \tag{3}$$

Additionally, $p_j \leq \rho_j$ *for all jobs and* $t_j \leq \rho_j$ *for all non-trivial jobs.*

Proof. Let j be any job in the instance. It is directly clear that

$$\tau_j = \min(t_j, u_j) \leq \min(t_j + p_j, u_j) = \rho_j,$$

since the processing times p_j are non-negative. The second inequality $\rho_j \leq p_j^A$ follows from the fact that ρ_j is the minimum of the two cases in the definition of p_j^A.

The first additional equation $p_j \leq \rho_j$ can be proven by a brief case distinction: If $\rho_j = t_j + p_j$, this follows from $t_j \geq 0$. If $\rho_j = u_j$, we have $p_j \leq u_j = \rho_j$ by the upper bound property (1). For the second additional equation, we note that since j is non-trivial, we have $t_j \leq u_j$ and thus $t_j = \tau_j \leq \rho_j$ by the already proven inequality (3). □

The above proposition arranges the running time variables by increasing size and introduces some helpful estimates for later use. The minimal running time τ_j is established as a lower bound for the running times of the offline optimum as well as any online algorithm. Since τ_j can be computed using offline input, this is incredibly helpful in designing methods, e.g. for sorting jobs or estimating parameters.

The proposition also states that the offline running time is always less than the algorithmic running time, which is very intuitive. At the same time, by employing certain testing strategies, we can guarantee that the algorithmic running time is not too large compared to the offline running time.

Recall the definition of $r_j = u_j/t_j$ from equation (2). The testing scheme in the following proposition was used frequently in both of our papers. It allows us to bound the algorithmic running times from above:

Proposition 5.4 ([6, Proposition 1, (b)(c)], [7, Proposition 1])**.** *Let an instance of scheduling with testing and an online algorithm ALG be given. Let ALG test a non-trivial job j if and only if $r_j \geq \alpha$ for some $\alpha \geq 1$. Then:*

(a) $\forall j \in [n]$, *j tested:* $p_j^A \leq \left(1 + \frac{1}{\alpha}\right) \rho_j$

(b) $\forall j \in [n]$, *j not tested:* $p_j^A \leq \alpha \rho_j$

Proof. First, assume the non-trivial job j is tested by ALG. Then it holds $r_j \geq \alpha$ as well as $p_j^A = t_j + p_j$. We again use a case distinction over the value of OPT: If $\rho_j = t_j + p_j$ as well, then clearly $p_j^A = \rho_j$. If, however, $\rho_j = u_j$, then

$$p_j^A = t_j + p_j \leq \frac{u_j}{\alpha} + u_j = \left(\frac{1}{\alpha} + 1\right) u_j = \left(1 + \frac{1}{\alpha}\right) \rho_j.$$

Here, we also used the upper bound property (1).

Now assume that j is not tested by ALG. Then we have $r_j < \alpha$ and $p_j^A = u_j$. In case that OPT behaves the same, we have $\rho_j = u_j$ and thus $p_j^A = \rho_j$. In the case that $\rho_j = t_j + p_j$, we get

$$p_j^A = u_j < \alpha t_j \leq \alpha(t_j + p_j) = \alpha \rho_j$$

by the non-negativity of p_j. □

Originally, the above proof concept was introduced in slightly different form by Dürr et al. [35]. They used a very similar line of arguments to prove that the simplified uniform testing scheme where α is fixed to a value of $\varphi \approx 1.6180$ leads to an optimal algorithm for the deterministic minimization of the makespan. In the first part of the makespan section below, we go into details on how their result follows from the above proposition and how the same approach can be used to prove the optimal randomized ratio of 4/3.

An additional insight is somewhat hidden in the proposition: Setting $\alpha = 1$ yields that all non-trivial jobs are tested and additionally

$$p_j^A \leq 2\rho_j, \qquad \forall j \in [n],$$

independent of the actual ratio r_j. For a single non-trivial job, testing and running it is therefore already 2-competitive, which holds for both objective functions. Comparatively, the algorithmic running time p_j^A can become arbitrarily large in comparison to ρ_j if a job is not tested.

5.2 Makespan

5.2.1 Optimal Results for a Single Machine

We start by considering the makespan objective on a single machine. We do not need to differentiate between different settings regarding preemption, since they are all equivalent.

The following two results were first proven by Dürr et al. [35] for uniform testing times $t_j \equiv 1$. The non-uniform versions were introduced in [6]. Recall that $\varphi \approx 1.6180$ is the golden ratio and that we defined $r_j = u_j/t_j$.

Theorem 5.5 ([35, Theorem 14], [6, Theorem 5]). *Let an instance of scheduling with testing on a single machine be given. The deterministic algorithm that tests all jobs j if and only if $r_j \geq \varphi$ is φ-competitive for the objective of minimizing the makespan. No deterministic algorithm can achieve a smaller competitive ratio.*

Theorem 5.6 ([35, Theorem 15], [6, Theorem 6]). *Let an instance of scheduling with testing on a single machine be given. The randomized algorithm that tests all jobs j with probability $p = 1 - 1/(r_j^2 - r_j + 1)$ is 4/3-competitive for the objective of minimizing the makespan. No randomized algorithm can achieve a smaller competitive ratio.*

Dürr et al. proved both results by reducing worst case instances for the makespan objective to a single job and then used a case distinction very similar to the proof of Proposition 5.4.

We now show how to use Proposition 5.4 directly to prove Theorem 5.5. We omit the proof for Theorem 5.6 since it is very similar and will not be used in later proofs.

Proof of Theorem 5.5. Let ALG be the algorithm described in the theorem. Let $j \in [n]$ be any job. If j is tested by ALG, we have by Proposition 5.4 and the testing scheme that

$$p_j^A \leq \left(1 + \frac{1}{\varphi}\right) \rho_j.$$

If j is not tested by the algorithm, we have by the same reasoning that

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

The defining property of the golden ratio is that $\varphi = 1 + 1/\varphi$. Therefore: $p_j^A \leq \varphi \rho_j$ in both cases.

To compare the values of OPT and ALG, we now only need to use Lemma 5.1:

$$\text{ALG} = \sum_{j=1}^{n} p_j^A \leq \sum_{j=1}^{n} \varphi \rho_j = \varphi \, \text{OPT}$$

It follows that ALG is φ-competitive. For the lower bound, we reference the corresponding proof in the complete version of Dürr et al. [34]. □

The above proof also reflects why the golden ratio corresponds to the optimal ratio in the one machine makespan setting: it minimizes the maximum of the two factors from Proposition 5.4. In the subsequent sections, we will often reference the case $\alpha := \varphi$ in Proposition 5.4 as the so-called *optimal strategy for a single job*. It always follows that $p_j^A \leq \varphi \rho_j$ for the corresponding job j.

5.2.2 Non-preemptive Results on Multiple Machines

We now consider the makespan objectives on m machines with $m \in \mathbb{N}$. Here, the algorithm has to determine where to assign jobs in addition to the testing decision. In this section, we always assume that the jobs are sorted by non-increasing value of the offline running times $\rho_1 \geq \cdots \geq \rho_n$. For now, we do not allow preemption, meaning that jobs cannot be interrupted and have to be executed on exactly one machine.

Bounds for the Value of the Optimum. Before we present our algorithms, we introduce an important concept for makespan minimization on multiple machines that is used frequently to prove competitive ratios.

The ultimate goal of any proof is to bound the value of ALG against the value of OPT from below. However, the value of OPT is often difficult to accurately calculate. Instead, we use so-called *lower bounds*, which suffice to guarantee competitiveness without the need to calculate the value of OPT explicitly. We give three very important lower bounds in the next few paragraphs.

The first bound is based on the explicit value of $\text{OPT} = \sum_{j=1}^{n} \rho_j$ on a single machine as given by Lemma 5.1. We extend this formula to more than one machine by taking

the average over all machines. Since the objective value corresponds to the *largest* completion time on any machine, the average is always at most OPT.

$$\frac{1}{m}\sum_{j=1}^{n}\rho_j \leq \mathrm{OPT} \tag{4}$$

The next lower bound formalizes the condition that the optimum has to schedule every job somewhere:

$$\rho_j \leq \mathrm{OPT}, \qquad \forall j \in [n] \tag{5}$$

Finally, we observe that if there are at least $m+1$ jobs in the instance, then some machine has to schedule at least two of them. Since the jobs are sorted by non-increasing offline running times, we get

$$\rho_m + \rho_{m+1} \leq \mathrm{OPT}\,. \tag{6}$$

Here, we define $\rho_k = 0$ in case the instance has less than k jobs.

The Extended List Scheduling Algorithm. The famous List Scheduling algorithm by Graham [51] schedules the current job on the least loaded machine. Here, the *load* is the sum of all processing times of jobs already assigned to the machine. We now combine this idea with the optimal testing strategy for a single job from Proposition 5.4 and Theorem 5.5.

The *Extended List Scheduling* algorithm works as follows: Consider the jobs in any order. For the next job j to be scheduled, test j if and only if $r_j \geq \varphi$ and assign it to the machine which has the current smallest load.

Theorem 5.7 ([7, Theorem 2])**.** *Let an instance of scheduling with testing on m machines be given. The Extended List Scheduling algorithm is $\varphi\left(2-\frac{1}{m}\right)$-competitive for minimizing the makespan. Extended List Scheduling can have no smaller competitive ratio.*

In the worst case, the algorithm has a competitive ratio of 3.2361. As we can see, the result directly combines the competitive ratios of the optimal strategy for a single job (φ) and the List Scheduling algorithm ($2-1/m$). In the proof, we combine Proposition 5.4 with the proof structure of List Scheduling. For this, we also use the lower bounds (4) and (5). Finally, we provide an example with many small jobs followed by one large job, showing that the algorithm cannot achieve a better ratio.

The SBS Algorithm. Our main result in paper [7] is an algorithm that beats Extended List Scheduling on $m \geq 2$ machines. It works in three phases named S_1, B, and S_2 and has a competitive ratio of 3.1016 in the worst case.

To describe the algorithm, we define a *threshold function* $T(m)$ depending on the number of machines m which is used for partitioning the job set $[n]$. Its definition is given further below in equation (7). Algorithm 1 provides the pseudo-code for the SBS algorithm.

Algorithm 1: SBS algorithm [7, Algorithm 1]

$B \leftarrow \{j \in [n] : r_j \geq T(m)\}$;
$S \leftarrow [n] \setminus B$;
$S_1 \leftarrow S' \subset S$, s.t. $|S'| = \min(m, |S|)$, $\tau_{j_1} \geq \tau_{j_2}$, $\forall j_1 \in S', j_2 \in S \setminus S'$;
$S_2 \leftarrow S \setminus S_1$;
foreach $j \in S_1$ **do**
    if $r_j \geq \varphi$ **then**
        test and run j on an empty machine;
    else
        run j untested on an empty machine;
    end
end
foreach $j \in B$ **do**
    test and run j on the current least-loaded machine;
end
foreach $j \in S_2$ **do**
    run j untested on the current least-loaded machine;
end

The threshold divides the jobs into a set of jobs $B = \{j \in [n] : r_j \geq T(m)\}$ with large parameter ratio and a set of jobs $S = [n] \setminus B$ with small parameter ratio. The set S is then further partitioned into S_1 and S_2, where S_1 contains the largest jobs in S with respect to the minimal running times and S_2 contains the remaining jobs.

The SBS algorithm first runs the at most m jobs in S_1 using the optimal testing strategy for single jobs. Then it runs all jobs in B untested, before testing and executing the remaining jobs from S_2.

The general idea of dividing the jobs into sets S and B is based on Proposition 5.4, where α is replaced by the threshold function $T(m)$. The reason for further subdividing S is that we want to use the lower bound (6) on the jobs in S_2. For this, we need to compare them against at least m other jobs which should be as large as possible. Since the minimal running time can be computed with offline input and S_1 contains by definition the m jobs with largest minimal running time, it is a natural choice for estimating the jobs in S_2.

The threshold function is kept general in the proof of the ratio of the algorithm and then optimized in the last step. The final definition of $T(m)$ is as follows:

$$T(m) = \frac{(3+\sqrt{5})m - 2 + \sqrt{(38+6\sqrt{5})m^2 - 4(11+\sqrt{5})m + 12}}{6m-2} \tag{7}$$

The resulting competitive value of the algorithm is also a function in m which we call $c(m)$. Its explicit form is given below:

$$c(m) = \frac{(3+\sqrt{5})m - 2 + \sqrt{(38+6\sqrt{5})m^2 - 4(11+\sqrt{5})m + 12}}{4m}$$

For explicit evaluations of the function, behavior for small and large values of m, and comparison to other results we refer to Chapter 4. It remains to state the theorem establishing this competitive ratio for the SBS algorithm.

Theorem 5.8 ([7, Theorem 3])**.** *Let an instance of scheduling with testing on m machines be given. The SBS algorithm is $c(m)$-competitive for minimizing the makespan.*

Proof sketch. Let us first consider the testing strategy of the SBS algorithm for a single job $j \in [n]$. Depending on which set of S_1, B, S_2 the job belongs to, its algorithmic running time may vary. By Proposition 5.4, we have

$$p_j^A \leq \begin{cases} \varphi\, \rho_j & \text{if } j \in S_1, \\ \left(1 + \frac{1}{T(m)}\right) \rho_j & \text{if } j \in B, \\ T(m)\, \rho_j & \text{if } j \in S_2. \end{cases} \tag{8}$$

Now let l be the index of the job that has the largest completion time under the produced schedule, i.e. the job that is responsible for the final value of the makespan $\max_j C_j$. Also let t denote the minimum load over all machines at the time just before ALG assigned job l to some machine. Clearly, we have

$$\mathrm{ALG} = \max_j C_j = C_l = t + p_l^A.$$

The remainder of the proof consists of estimating $t + p_l^A$ depending on which set job l belongs to. The mathematical details are given by the following proposition:

Proposition 5.9 ([7, Proposition 2])**.** *The value of the algorithm can be estimated as follows:*

$$\mathrm{ALG} \leq \begin{cases} \varphi\, \mathrm{OPT} & \text{if } l \in S_1, \\ \left(\varphi + \left(1 + \frac{1}{T(m)}\right)\left(1 - \frac{1}{m}\right)\right) \mathrm{OPT} & \text{if } l \in B, \\ T(m) \left(\frac{3}{2} - \frac{1}{2m}\right) \mathrm{OPT} & \text{if } l \in S_2. \end{cases}$$

A proof is provided in the full version [8] of paper [7]. It combines the lower bounds (4) - (6) with the estimates (8) for the algorithmic running time of a job under the SBS algorithm. As we have pointed out, the critical estimation of p_l^A for $l \in S_2$ is possible since the algorithm has already scheduled m larger jobs w.r.t. the minimal running time from S_1 beforehand. In addition, we also need inequality (3) from Proposition 5.3 for the estimation of the minimal running times in terms of the optimum.

For the final ratio, we take the maximum over the three cases in Proposition 5.9 and minimize this value in dependence of $T(m)$. The case $l \in S_1$ is always smaller than the other cases. It can be easily checked that the definition (7) of $T(m)$ minimizes the maximum of the two remaining cases for all values $m \geq 1$. Similarly, it can be verified that the value of the resulting maximum is given by $c(m)$ and thus $\mathrm{ALG} \leq c(m)\, \mathrm{OPT}$. □

The Uniform-SBS Algorithm. If the scheduling with testing instance is *uniform*, then all testing times t_j are equal to 1. For this slightly easier setting, we present an improved version of the SBS algorithm called Uniform-SBS, which is 3-competitive in the worst case. For this, we need an alternate uniform definition of the threshold:

$$T_1(m) = \frac{2m - 1 + \sqrt{16m^2 - 14m + 3}}{3m - 1}$$

Uniform-SBS then works as follows: Sort all jobs by non-increasing upper bound value u_j. Go through the sorted list one by one and schedule the next job on the machine which has the current lowest load. Here, the load is again the sum of all processing times of jobs already assigned to the machine. Test job j if and only if $u_j \geq T_1(m)$.

The competitive ratio of Uniform-SBS is given by $c_1(m)$, where

$$c_1(m) = \frac{2m - 1 + \sqrt{16m^2 - 14m + 3}}{2m}.$$

We again refer to Chapter 4 for details on the behavior of this function and comparison to other results.

Theorem 5.10 ([7, Theorem 4])**.** *Let an instance of scheduling with testing on m machines be given where all testing times are uniformly equal to 1. The Uniform-SBS algorithm is $c_1(m)$-competitive for minimizing the makespan.*

We omit the proof and only give a brief intuition here: If all testing times are equal to 1, then sorting by u_j is equivalent to sorting by the parameter ratio $r_j = u_j/1$. Hence, Uniform-SBS first tests and runs all jobs with a large ratio and afterwards executes all jobs with a small ratio untested. Therefore, Uniform-SBS corresponds to the non-uniform SBS algorithm without the extra division of small jobs into the sets S_1 and S_2. In the proof, we again use the lower bounds (4) - (6) to estimate the algorithmic value of the algorithm. In particular, we can use (6) without the set S_1 of the non-uniform version, since the fixed testing times $t_j \equiv 1$ replace the role of the minimal running times.

Lower Bound. The following lower bound is based on a typical counterexample for List Scheduling where many small jobs are followed by a single large job. If we let the upper bounds be very large in comparison to the testing times, then an algorithm must test every job, forcing it to decide on a machine with almost no information about the real processing time of the job. The result holds even in the case of uniform testing times. The proof is omitted.

Theorem 5.11 ([7, Theorem 1])**.** *Let an instance of scheduling with testing on m machines be given where all testing times are uniformly equal to 1. In the non-preemptive setting, no online algorithm for minimizing the makespan is better than $\max(\varphi, 2-1/m)$-competitive.*

An Improved Result for Uniform Instances with Few Non-trivial Jobs. Recall that a job was called *trivial* if $t_j > u_j$. Such jobs are never tested by any reasonable algorithm. We now present a minor result that works only for uniform instances where the number of non-trivial jobs is at most m.

Lemma 5.12 ([8, Lemma 1])**.** *Let an instance of scheduling with testing on m machines be given where all testing times are uniformly equal to 1 and the number of non-trivial jobs is at most m. There exists a $\varphi\left(\frac{4}{3} - \frac{1}{3m}\right)$-competitive online algorithm for minimizing the makespan.*

In the worst case, the above competitive ratio is 2.1574. The corresponding algorithm first distributes all non-trivial jobs over all machines, which is possible since there are at most m such jobs. Tests are decided by the optimal testing strategy for a single job. The remaining jobs are trivial, therefore the algorithm has full knowledge about the relevant parameters and can employ the offline algorithm *Largest Processing Time first* (LPT) [52].

We note that the lower bound presented in the previous section is based on a counterexample with more than $m(m-1)$ jobs and is therefore not applicable to this setting.

5.2.3 Preemptive Results on Multiple Machines

We are still studying the makespan objectives on m machines with $m \in \mathbb{N}$. In this section, we now allow preemption, meaning jobs can be interrupted and then executed on a different machine after the interruption. We consider both the *test-preemptive* and the ordinary *preemptive* setting in accordance with the definitions we gave in Chapter 2.

The Two Phases Algorithm. For the test-preemptive setting, we provide a straightforward algorithm called *Two Phases*. The algorithm can be applied without changes to the slightly easier ordinary preemptive setting. It works as follows:

Let OFF be any optimal offline algorithm, e.g. a brute force method. In the first phase, apply OFF to the offline instance consisting of all jobs with processing requirements equal to their minimal running time τ_j. Afterwards, all trivial jobs are completely executed, while the non-trivial jobs have all been tested. Therefore all processing times p_j of the remaining jobs are known. In the second phase, apply OFF again, this time to the offline instance consisting of all remaining jobs with processing requirements equal to p_j. Put the resulting schedule obliviously on top of the schedule produced in phase one.

Theorem 5.13 ([7, Theorem 5])**.** *Let an instance of scheduling with testing on m machines be given. The Two Phases algorithm is 2-competitive for minimizing the makespan in the test-preemptive setting.*

In accordance with Definition 2.1, we may assume that an online algorithm has access to unlimited computational power. Therefore, it is valid to use for example a brute force method as the offline algorithm OFF. In case we do not allow a non-polynomial

runtime of the algorithm, we can instead use the PTAS by Hochbaum and Shmoys [56], to achieve a competitive ratio of $2 + \varepsilon$ for any $\varepsilon > 0$.

The proof consists of two straightforward applications of OFF in combination with Proposition 5.3 and is omitted here.

Lower Bound. We were able to prove a lower bound that holds for the ordinary preemptive setting and thus can be applied to the test-preemptive setting, too. Combined with the Two Phases algorithm above, we have therefore almost completely solved the (test)-preemptive setting for large values of m. The results also holds for uniform testing times.

Theorem 5.14 ([7, Theorem 6])**.** *Let an instance of scheduling with testing on m machines be given where all testing times are uniformly equal to 1. In the preemptive setting, no online algorithm for minimizing the makespan is better than $\max(\varphi, 2 - 2/m + 1/m^2)$-competitive.*

The proof is omitted. It is similar to the proof of Theorem 5.11.

5.2.4 Fully-Online Results on Multiple Machines

As the final portion of results for the makespan objective, we consider the fully-online variant. Jobs are presented to the algorithm one by one, and it does not know the total number of jobs. Whenever a job arrives, its upper bound and its testing time become known to the algorithm. Only if the job is tested, its processing time is revealed. After completely and irrevocably assigning a job to some machine, the next one is presented to the algorithm. We only examine results for the non-preemptive setting.

Extended List Scheduling Applied to the Fully-Online Case. The Extended List Scheduling algorithm we provided in Theorem 5.7 is $\varphi(2 - 1/m)$-competitive. It can be applied to the fully-online case without any changes: Since the algorithm did not impose any order on the jobs, it still works when they arrive one by one.

Corollary 5.15. *Let an instance of scheduling with testing on m machines be given. The Extended List Scheduling algorithm is $\varphi\left(2 - \frac{1}{m}\right)$-competitive for minimizing the makespan in the fully-online setting.*

Proof. This follows directly from Theorem 5.7 and the fact that Extended List Scheduling is a fully-online algorithm considering jobs one by one. □

Lower Bounds. It is clear that the lower bound from Theorem 5.11 can be extended to the fully-online variant. For $m = 1$, the Extended List Scheduling had the same ratio as this lower bound, hence the single machine case remains completely solved when jobs arrive one by one. We now present a slightly better lower bound for $m \geq 2$ than we were able to prove for the ordinary version.

Theorem 5.16 ([8, Theorem 7]). *Let an instance of scheduling with testing on $m \geq 2$ machines be given where all testing times are uniformly equal to 1. In the fully-online setting, no online algorithm for minimizing the makespan is better than 2-competitive.*

The proof consists of a simple counterexample with $m + 1$ jobs.

To conclude the section on the fully-online makespan objective, we provide a final lower bound that is even larger than 2. In the proof, we use numerical parameter optimization with Mathematica [86]. Because the number of necessary parameters in the proof increases exponentially with m, we were unable to prove this result for a general number of machines and present the simplest case $m = 2$ as a substitute. However, we conjecture that the result can indeed be extended to any number of machines larger than two.

Theorem 5.17 ([8, Theorem 8]). *Let an instance of scheduling with testing on $m = 2$ machines be given. In the fully-online setting, no online algorithm for minimizing the makespan is better than 2.0953-competitive.*

5.3 Sum of Completion Times

5.3.1 Deterministic Test-Preemptive Results for a Single Machine

We now consider the sum of completion times objective on one machine in the deterministic test-preemptive setting. Jobs can be interrupted, but only right after their test has ended. Untested jobs cannot be interrupted. The algorithm has to determine the order of the tests and executions in addition to the testing decision. In this section, we again assume that the jobs are sorted by non-increasing value of the offline running times $\rho_1 \geq \ldots \geq \rho_n$.

Comparison to the Uniform Case. Dürr et al. [35] showed that there exists a 2-competitive algorithm for the deterministic test-preemptive setting. They decide testing and delaying of jobs based on a threshold value of 2. In this context, *delaying* a job means to interrupt it right after its test and then continue it at some point later in the schedule.

We note here that Dürr et al. later gave an even easier 2-competitive algorithm in the journal version of their paper [34] and analyzed it using their methods. In the appendix of our full version [5], we summarize how our approach for the general case can be applied to provide an alternative proof of this result.

One important insight by Dürr et al. was the following lemma concerning jobs with small upper bounds:

Lemma 5.18 ([35, Lemma 1]). *Let an instance of scheduling with testing on one machine be given where all testing times are uniformly equal to 1. Without loss of generality, we can assume that any c-competitive algorithm for minimizing the sum of completion times executes all jobs with $u_j < c$ untested in non-decreasing order of u_j*

in the beginning of the schedule. In particular, worst case instances for the algorithm consist only of jobs j with $u_j \geq c$.

The approach in the proofs of the competitive ratios in [35] was to describe the worst case ratio by a parameterized formula. For this, they relied on Lemma 5.18 to reduce worst case instances to a simple structure, which they could then describe with such a formula. In the full version [5], we show that the lemma cannot be generalized to non-uniform tests, signifying the necessity of a new approach to solve these more difficult instances.

It is clear that Lemma 5.18 does not generalize directly: a single job with $0 < u_j < c$, $p_j = t_j = 0$ has to be tested to guarantee a finite competitive ratio. Instead, it seems intuitive to generalize the lemma based on the parameter ratio $r_j = u_j/t_j$. However, it can be shown [5] that for any $c \geq 1$ executing all jobs with $r_j < c$ untested in the beginning of the schedule leads to an arbitrarily bad outcome. The proof is based on a counterexample with one large job and many small jobs. If we let the size of the large job and the number of small jobs approach infinity at the same time, the ratio between any algorithm obeying the rule that jobs with $r_j < c$ are run untested in the beginning and the optimum becomes arbitrarily large.

The problem with running jobs with small parameter ratio first is that potentially large jobs are pushed to the front of the schedule and cause high completion times for all following jobs. Hence, an algorithm for the non-uniform case must carefully consider the size of all parameters, not only during the testing decision, but also when sorting the executions. Our main algorithm (α, β)-SORT takes into account not only the job configurations, but also the resulting execution lengths when deciding on the order of the jobs.

The (α, β)-SORT Algorithm. We now present the main result from our paper [6]: a 4-competitive algorithm for the deterministic test-preemptive setting. The basic premise is similar to previous algorithms we already presented: We let the parameter $\alpha \geq 1$ correspond to the threshold in Proposition 5.4 and use it to partition the jobs into two sets $B = \{j \in [n] : r_j \geq \alpha\}$ and $S = [n] \setminus B$, where jobs from B are tested and jobs from S are not. The new parameter $\beta \geq 1$ is used for sorting the jobs in an adequate way: the next job scheduled on the machine is determined by the current smallest *scaling time* σ_j, where

$$\sigma_j = \begin{cases} u_j & \text{if } j \in S, \\ \beta t_j & \text{if } j \in B \text{ and has not been tested,} \\ p_j & \text{if } j \in B \text{ and has already been tested.} \end{cases}$$

As we can see, β is used for artificially increasing the testing time of a job $j \in B$, such that it is considered later than other executions in the ordering of the algorithm. Intuitively, testing can be considered less important, since it does not immediately lead to a job being completed. We provide the complete pseudo-code for (α, β)-SORT in Algorithm 2.

Algorithm 2: (α, β)-SORT [6, Algorithm 1]

```
B ← {j ∈ [n] : r_j ≥ α};
S ← [n] \ B;
foreach j ∈ B do
    σ_j ← βt_j;
end
foreach j ∈ S do
    σ_j ← u_j;
end
while B ∪ S ≠ ∅ do
    choose j_min ∈ argmin_{j∈B∪S} σ_j;
    if j_min ∈ S then
        run j_min untested;
        remove j_min from S;
    else if j_min ∈ B then
        if j_min not tested then
            test j_min;
            σ_{j_min} ← p_{j_min};
        else
            run j_min;
            remove j_min from B;
        end
    end
end
```

We keep α and β general and optimize them in the final step of the proof. Keeping the parameters general in the algorithm and the proof is helpful to better demonstrate the structural ideas behind them. Moreover, the algorithm framework and some of the achievements in the proof will be reused later in the results of Sections 5.3.2 and 5.3.4 with different values for α and β.

We achieved a best possible competitive ratio of 4 in the (α, β)-SORT algorithm by setting $\alpha = \beta = 1$. Choosing the parameters like this is slightly at odds with the intuition behind them: a value of $\alpha = 1$ means that all non-trivial jobs are tested by the algorithm, which is likely not an optimal strategy in view of the makespan results presented earlier. Setting $\beta = 1$ signifies that other executions are not prioritized over testing, putting the impact of this parameter into question. However, in the randomized setting, the optimal choice for β will in fact be larger, hence the fundamental idea behind the parameter is indeed valid. In summary, we conjecture that an improved bound is possible by choosing other values for the parameters in (α, β)-SORT. However, for the closed coefficient formula we received through our proof, setting $\alpha = \beta = 1$ was optimal.

To further investigate how much the algorithm can be improved by other parameter choices, we considered algorithmic lower bounds in the full version [5] of our paper.

We proved that for any $\alpha, \beta \geq 1$, the algorithm cannot be better than 2-competitive. Additionally, $(1,1)$-SORT itself can have no competitive ratio smaller than 3.

Theorem 5.19 ([6, Theorem 1])**.** *Let an instance of scheduling with testing on one machine be given. The $(1,1)$-SORT algorithm is 4-competitive for minimizing the sum of completion times in the test-preemptive setting.*

Proof sketch. To compute the objective value of an algorithm, we need to estimate the completion times of all jobs (c.f. Definition 2.4). For this, we introduce the notion of *contributions*: We define the contribution $c(k,j)$ of job k to the completion time C_j of job j as the amount of time on the schedule that the algorithm assign to k before the point where j is completed. The contribution of k to j is clearly upper bounded by the total amount of time the algorithm spends on scheduling k, or formally: $c(k,j) \leq p_k^A$. The following lemma describes the completion time of a job j in terms of the contributions of other jobs k. It additionally gives a very useful estimation of $c(k,j)$ independent of k.

Lemma 5.20 (Contribution Lemma, [6, Lemma 1])**.** *Let $j \in [n]$ be a given job. The completion time of j can be written as*

$$C_j = \sum_{k=1}^{n} c(k,j). \tag{9}$$

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

$$c(k,j) \leq \max\left(\left(1+\frac{1}{\beta}\right)\alpha, 1+\frac{1}{\alpha}, 1+\beta\right)\rho_j. \tag{10}$$

Expressing C_j in terms of the contributions is straightforward. The estimation in the second part of the lemma relies heavily on the inequalities from Proposition 5.3 and the testing scheme from Proposition 5.4. The complete proof is provided in the full version [5] of paper [6].

In the estimation for C_j, we will use inequality (10) from the Contribution Lemma to bound the contribution of jobs $k \leq j$, and use $c(k,j) \leq p_k^A$ directly to bound the contribution of jobs $k > j$. We have

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

where we also used Proposition 5.4 once more in the final line.

Now we can estimate the algorithmic value against the optimum. Since we made the assumption $\rho_1 \geq \ldots \geq \rho_n$ in the beginning of the section, we can use Lemma 5.2. Therefore:

$$\begin{aligned}
\text{ALG} &= \sum_{j=1}^{n} C_j \\
&\leq \sum_{j=1}^{n} \sum_{k=j+1}^{n} \max\left(\alpha, 1+\frac{1}{\alpha}\right) \rho_k + \sum_{j=1}^{n} \max\left(\left(1+\frac{1}{\beta}\right)\alpha, 1+\frac{1}{\alpha}, 1+\beta\right) j \cdot \rho_j \\
&= \max\left(\alpha, 1+\frac{1}{\alpha}\right) \sum_{j=1}^{n} (j-1)\rho_j + \max\left(\left(1+\frac{1}{\beta}\right)\alpha, 1+\frac{1}{\alpha}, 1+\beta\right) \sum_{j=1}^{n} j \cdot \rho_j \\
&\leq \left(\max\left(\alpha, 1+\frac{1}{\alpha}\right) + \max\left(\left(1+\frac{1}{\beta}\right)\alpha, 1+\frac{1}{\alpha}, 1+\beta\right)\right) \text{OPT}
\end{aligned}$$

Minimizing the coefficient of this final estimation in dependence of α and β leads to an optimal value of 4 when setting $\alpha = \beta = 1$. □

5.3.2 Deterministic Preemptive Results for a Single Machine

This section highlights an algorithm for the sum of completion times on one machine in the deterministic preemptive setting. Now, jobs can be interrupted at any time during their execution, independent of their configuration. The algorithm has to determine where to interrupt jobs, how to order all execution fragments, and of course decide whether a job is tested or not. As before, we assume that the jobs are sorted by non-increasing value of the offline running times $\rho_1 \geq \ldots \geq \rho_n$.

The Golden Round Robin Algorithm. The basic idea for our algorithm in the ordinary preemptive setting stems from the so-called *Round Robin* rule. Pinedo [73] describes this common approach in preemptive machine scheduling as follows: Round Robin continually cycles through the list of all jobs, tending to each job for a fixed, very small time unit, before switching to the next job. It ensures that, at any time, any single job has received at most one unit of processing time more than any other job.

For the scheduling with testing setting, we combine this method with our optimal strategy for a single job. Below, we give a complete description of the resulting *Golden Round Robin* algorithm, which is 3.2361-competitive in the worst case:

First, decide the configuration for all jobs: If $r_j \geq \varphi$, test the job, if not, run it untested. Then, run all jobs in a Round Robin scheme until every job has been completely executed. Whenever a job has been run for a total time equal to p_j^A, i.e. it is completely executed, drop it from the rotation and continue the Round Robin with the remaining jobs.

For jobs that the algorithm wants to execute tested, the actual value of p_j^A is not known until the corresponding test is completed. Therefore, it is important to note that the Round Robin scheme works even under incomplete knowledge of the processing times of the jobs, as long as it knows when a job will be completed [73].

Theorem 5.21 ([6, Theorem 2])**.** *Let an instance of scheduling with testing on one machine be given. The Golden Round Robin algorithm is* 2φ*-competitive for minimizing the sum of completion times in the preemptive setting. Golden Round Robin can have no smaller competitive ratio.*

In the proof of this theorem, we again use Proposition 5.4 towards the optimal strategy for a single job. To estimate the completion time C_j, we categorize the contributions of other jobs based on whether they finish before or after j in the Round Robin scheme. To show that the algorithm cannot achieve a better competitive ratio, we provide a sequence of examples where the algorithmic objective value approaches $2\varphi\,\mathrm{OPT}$.

Lower Bound. Since Dürr et al. [35] did not consider the ordinary preemptive case, we provide a dedicated lower bound for this case which builds on the test-preemptive lower bound from their paper. Therefore, it even holds for uniform testing times.

Theorem 5.22 ([6, Theorem 3])**.** *Let an instance of scheduling with testing on one machine be given where all testing times are uniformly equal to* 1*. In the preemptive setting, no online algorithm for minimizing the sum of completion times is better than* 1.8546*-competitive.*

In the proof, we adopt the counterexample given by [35] and show that any preemptive algorithm on this instance can be replaced by a test-preemptive algorithm with the same competitive ratio. Since no test-preemptive result better than 1.8546 exists, the same follows for the preemptive case.

5.3.3 Deterministic Non-preemptive Impossibility Result

In this section we show via a counterexample that no finite competitive ratio is possible in the strictly non-preemptive setting on one machine. This result is new in this thesis and was not part of any prior publications.

Theorem 5.23. *Let an instance of scheduling with testing on one machine be given where all testing times are uniformly equal to* 1*. In the non-preemptive setting, no online algorithm for minimizing the sum of completion times can have a finite competitive ratio.*

Proof. Let $n \in \mathbb{N}$ jobs be given, and let $M \in \mathbb{N}$ be defined as $M = n^3$. Let every job have $u_j = M, t_j = 1$. Depending on the choices of a given algorithm ALG, an adversary will determine the value of the p_j.

Consider first the case that ALG does not test every job, i.e. there is at least one job k that is run untested. The adversary sets $p_j = 0,\ \forall j \in [n]$, and therefore by Lemma 5.2 we have

$$\mathrm{OPT} = \sum_{j=1}^{n} j \cdot \rho_j = \sum_{j=1}^{n} j = \frac{n^2 + n}{2}.$$

The algorithm runs k untested and therefore

$$\mathrm{ALG} = \sum_{j=1}^{n} C_j = C_k + \sum_{j \neq k} C_j \geq M + \frac{n^2 - n}{2}.$$

If we let $n \to \infty$, we have $\mathrm{ALG} / \mathrm{OPT} \to \infty$ due to $M = n^3$.

Now consider the case that all jobs are tested. Since the upper bounds are quite high, this is in fact the expected behavior of ALG. We will see that the competitive ratio can still become arbitrarily bad.

W.l.o.g. we may assume that ALG tests the jobs in order of their index since they are indistinguishable. Then the adversary sets all job processing times to $p_j = 0$, except for the first job $j = 1$ for which it sets $p_1 = M$. It follows that the optimal offline running times ρ_j are sorted by non-increasing size and therefore by Lemma 5.2:

$$\mathrm{OPT} = \sum_{j=1}^{n} j \cdot \rho_j = M + \sum_{j=2}^{n} j = M + \frac{n^2 + n}{2} - 1$$

The algorithm, however, is forced by the adversary to schedule the large job $j = 1$ first on the machine. Since all jobs have to wait for this first job to finish, this leads to very large completion times $C_j = M + j$ for *all* $j \in [n]$. In total:

$$\mathrm{ALG} = \sum_{j=1}^{n} C_j = \sum_{j=1}^{n} (M + j) = n \cdot M + \frac{n^2 + n}{2}$$

Again, we let $n \to \infty$. Since $M = n^3$, it follows again that $\mathrm{ALG} / \mathrm{OPT} \to \infty$. □

As we can see, the problem in the non-preemptive sum of completion times objective is that the algorithm cannot prevent a large job to occur in the beginning of the schedule. This causes all subsequent jobs to have a large completion time as well and leads to a very high competitive ratio.

5.3.4 Randomized Test-Preemptive Results for a Single Machine

In the final section of this chapter, we consider the sum of completion times objective on one machine in the *randomized* test-preemptive setting. An algorithm has to determine the order of the jobs and decide the testing, but it now also has access to a number of random choices during its runtime. Jobs can be interrupted, but only right after their test has ended. Untested jobs cannot be interrupted. Once again, we assume that the jobs are sorted by non-increasing value of the offline running times $\rho_1 \geq \ldots \geq \rho_n$.

The randomized algorithm *Random-SORT* we present is very similar to the deterministic (α, β)-SORT. We replace the testing scheme based on Proposition 5.4 by a randomized decision of whether a job is tested or not. The probability $p(r_j)$ of job j being tested depends on the parameter ratio r_j and is done independently for all jobs. We note that trivial jobs are exempt from this and are never tested. The sorting parameter β is adopted without changes. We give the pseudo-code in Algorithm 3.

Theorem 5.24 ([6, Theorem 4])**.** *Let an instance of scheduling with testing on one machine be given. The Randomized-SORT algorithm is* 3.3794*-competitive for minimizing the sum of completion times in the test-preemptive setting.*

Algorithm 3: Randomized-SORT [6, Algorithm 3]

```
B ← ∅, S ← ∅;
foreach j ∈ [n] do
    if r_j ≥ 1, add j to B with probability p(r_j) and set σ_j ← βt_j;
    otherwise add j to S and set σ_j ← u_j;
end
while B ∪ S ≠ ∅ do
    choose j_min ∈ argmin_{j∈B∪S} σ_j;
    if j_min ∈ S then
        run j_min untested;
        remove j_min from S;
    else if j_min ∈ B then
        if j_min not tested then
            test j_min;
            σ_{j_min} ← p_{j_min};
        else
            run j_min;
            remove j_min from B;
        end
    end
end
```

The function $p(r_j)$ and the parameter β are kept general in the proof and are then optimized using the mathematical software Mathematica [86]. The final value of β will be approximately 1.2574. The final choice of $p(r_j)$ fulfills $p(1) = 0$ and increases in r_j. For $r_j > 2.1637$, it holds that $p(r_j) = 1$, hence jobs with large parameter ratio are always tested. The function term is too cumbersome to explicitly state here, thus we refer to the corresponding proof in paper [6] and its full version [5]. Since Randomized-SORT is the only elaborate randomized algorithm that is studied in this thesis, we give a sketch of the main proof ideas below.

Proof sketch. Based on the randomized testing scheme, the algorithmic running times are now random variables and it holds

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

independently for all jobs j.

Similarly, the contributions $c(k, j)$ for fixed jobs j and k are now also random variables. We estimate the values for all jobs with Propositions 5.3 and 5.4, and then use the law of total expectation to compute the expected value:

$$\mathbb{E}[c(k,j)] \leq \left(1 + \frac{1}{\beta}\right) u_j \cdot (1 - p(r_j)) + \max\left((1+\beta)t_j, \left(1 + \frac{1}{\beta}\right) p_j, t_j + p_j\right) \cdot p(r_j)$$

Alternatively, we may again use $\mathbb{E}[c(k,j)] \leq \mathbb{E}[p_k^A]$ directly for all jobs with large index $k > j$. We combine the bounds with equation (9) from the Contribution Lemma and obtain

$$\mathbb{E}[\text{ALG}] \leq \sum_{j=1}^{n} j \cdot \lambda_j(\beta, p(r_j)),$$

where $\lambda_j(\beta, p(r_j))$ is a function depending on job j, parameter β, and the probability $p(r_j)$. The final competitive ratio of Random-SORT will depend on the worst case ratio $\max_j \lambda_j(\beta, p(r_j))/\rho_j$ over all jobs of the instance. The following lemma establishes an upper bound for this value:

Lemma 5.25 ([6, Lemma 2])**.** *There exist a parameter $\hat{\beta} \geq 1$ and a probability function $\hat{p}(r_j) \in [0,1]$ such that*

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

The explicit choice for parameter β is $\hat{\beta} \approx 1.2574$. The somewhat lengthy term $\hat{p}(r_j)$ is given in the proof of the lemma, which we omit here. The proof includes computer-aided optimization with Mathematica [86], all details including the code can be found in the full version [5].

Finally, we estimate the value of the algorithm using the lemma above in combination with Lemma 5.2:

$$\mathbb{E}[\text{ALG}] \leq \sum_{j=1}^{n} j \cdot \lambda_j(\hat{\beta}, \hat{p}(r_j)) \leq 3.3794 \sum_{j=1}^{n} j \cdot \rho_j = 3.3794 \cdot \text{OPT}$$

□

6 Open Research Questions and Directions

This thesis presented algorithms and lower bounds for the scheduling with testing problem, a theoretical optimization framework that is concerned with scheduling applications where uncertain information can be obtain by investing some additional cost.

It is clear that the research area of scheduling with testing setting is far from completed, nor fully understood. Aside from the obvious gaps to be closed between the achieved algorithmic ratios and lower bounds, an abundance of other related directions are viable for future research. In this section, we talk about some of the missing results that we were unable to provide and then go over research possibilities for the future.

We begin by summarizing some of the shortcomings in our results regarding open gaps. For the sum of completion times, no dedicated lower bounds for general testing times that beat the uniform version could be found. As we have conjectured in [6], in order to achieve better lower bounds it seems the adversary has to modify the input during the runtime of an algorithm based on its decisions. For non-uniform testing times we did not find a way to describe such adversarial strategies in general. Additionally, the final choice of parameters in our main algorithm (α, β)-SORT was likely not optimal. It might be possible to find and prove better parameters using a different proof strategy under the same algorithmic framework.

For the makespan objective on more than one machine, there is also quite a large distance between the best algorithmic value and the lower bound we provided. In terms of an improvement of the algorithmic solution, we were unable to find general methods to beneficially sort jobs other than grouping them by testing configuration. We conjecture that some clever sorting scheme would improve upon the best possible competitive ratio. We also think it likely that some improved lower bound can be achieved by combining the typical construction from makespan minimization with the difficulty of the testing decision. However, it is tricky to achieve one of the two without sacrificing the other within a single counterexample.

In terms of possible directions, we first mention the promising variation we only considered very briefly in this thesis and our publications: Fully-online arrivals of jobs in the scheduling with testing setting. As we have seen in Section 5.2.4, this variation is at least as hard as the ordinary problem. Our Extended List Scheduling algorithm is a basic method which can be applied and provides a first upper bound. Closing the gap between this value and the simple lower bound we provided, as well as the introduction of customized methods and approaches are compelling directions for future research.

Furthermore, it could be interesting to consider other multi-machine models besides identical parallel machines. Results for related machines with different speeds often

contain the golden ratio in some way [75], which might hint to a possible methodological connection to scheduling with testing. Other interesting settings to be considered include unrelated machines or multi-stage job scheduling problems.

The golden ratio also appears frequently in ratios from algorithms for scheduling with rejections (see also Section 3.7). Some of the methods for this problem can be compared to approaches used in scheduling with testing, in particular concerning the balancing decisions an algorithm has to address. It could be promising to further study the connection between the two settings.

Finally, we mention possible variations of the underlying framework of our model. In certain applications, it might be plausible to consider slightly different parameter conditions. For example, one could study the setting where the processing time of a job j is *always* given by p_j, even in the case that the job is run untested, and u_j is effectively only a bound for this value. Dürr et al. [35] briefly mention a setting with additional lower limits l_j, where job processing times lie in an interval $[t_j, u_j]$. Though they point out that this setting is not harder than without lower limits, it might still be interesting to study how the additional information influences the outcome, for example in combination with general testing times. Other possible variants include the addition of *release times* for the sum of completion times objective, considering job-dependent *weights*, or examining other objective functions. Historically, explorable uncertainty was concerned with minimizing the number of queries instead of subtracting costs from an objective function. In scheduling with testing this could be incorporated for example with a dedicated machine that is only allowed to run tests. Numerous other possibilities are conceivable, each presenting new opportunities to explore and study.

Bibliography

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 32–43, 1999.

[2] S. Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29(2):459–473, 1999.

[3] S. Albers. On randomized online scheduling. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 134–143, New York, NY, USA, 2002. Association for Computing Machinery.

[4] S. Albers. Recent advances for a classical scheduling problem. In F. V. Fomin, R. Freivalds, M. Kwiatkowska, and D. Peleg, editors, *Automata, Languages, and Programming*, pages 4–14, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[5] S. Albers and A. Eckl. Explorable uncertainty in scheduling with non-uniform testing times. arXiv preprint, 2020.

[6] S. Albers and A. Eckl. Explorable uncertainty in scheduling with non-uniform testing times. In C. Kaklamanis and A. Levin, editors, *Approximation and Online Algorithms*, Lecture Notes in Computer Science, pages 127–142, Cham, 2021. Springer International Publishing.

[7] S. Albers and A. Eckl. Scheduling with testing on multiple identical parallel machines. In A. Lubiw and M. Salavatipour, editors, *Algorithms and Data Structures*, Lecture Notes in Computer Science, pages 29–42, Cham, 2021. Springer International Publishing.

[8] S. Albers and A. Eckl. Scheduling with testing on multiple identical parallel machines. arXiv preprint, 2021.

[9] S. Albers and M. Hellwig. Semi-online scheduling revisited. *Theoretical Computer Science*, 443:1–9, 2012.

[10] S. Albers and M. Hellwig. On the value of job migration in online makespan minimization. *Algorithmica*, 79(2):598–623, 2017.

[11] S. Albers and M. Hellwig. Online makespan minimization with parallel schedules. *Algorithmica*, 78(2):492–520, 2017.

[12] S. Albers and M. Janke. Online makespan minimization with budgeted uncertainty. In A. Lubiw and M. Salavatipour, editors, *Algorithms and Data Structures*, pages 43–56, Cham, 2021. Springer International Publishing.

[13] S. Anand, N. Garg, and A. Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *Proceedings of the 2012 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1228–1241, 2012.

[14] L. Arantes, E. Bampis, A. Kononov, M. Letsios, G. Lucarelli, and P. Sens. Scheduling under Uncertainty: A Query-based Approach. In *IJCAI 2018 - 27th International Joint Conference on Artificial Intelligence*, pages 4646–4652, Stockholm, Sweden, July 2018.

[15] Y. Azar and O. Regev. On-line bin-stretching. *Theoretical Computer Science*, 268(1):17–41, 2001.

[16] J. Balogh, J. Békési, G. Dósa, J. Sgall, and R. van Stee. The optimal absolute ratio for online bin packing. *Journal of Computer and System Sciences*, 102:1–17, 2019.

[17] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '92, pages 51–58, New York, NY, USA, 1992. Association for Computing Machinery.

[18] Y. Bartal, H. Karloff, and Y. Rabani. A better lower bound for on-line scheduling. *Information Processing Letters*, 50(3):113–116, 1994.

[19] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, and L. Stougie. Multiprocessor scheduling with rejection. *SIAM Journal on Discrete Mathematics*, 13(1):64–78, 2000.

[20] E. M. L. Beale. On minimizing a convex function subject to linear inequalities. *Journal of the Royal Statistical Society: Series B (Methodological)*, 17(2):173–184, 1955.

[21] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994.

[22] M. Böhm, J. Sgall, R. Van Stee, and P. Veselỳ. A two-phase algorithm for bin stretching with stretching factor 1.5. *Journal of Combinatorial Optimization*, 34(3):810–828, 2017.

[23] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[24] R. Bruce, M. Hoffmann, D. Krizanc, and R. Raman. Efficient update strategies for geometric computing with uncertainty. *Theory of Computing Systems*, 38(4):411–423, 2005.

[25] 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.

[26] M. Charikar, R. Fagin, V. Guruswami, J. Kleinberg, P. Raghavan, and A. Sahai. Query strategies for priced information. *Journal of Computer and System Sciences*, 64(4):785–819, 2002.

[27] B. Chen, A. van Vliet, and G. J. Woeginger. A lower bound for randomized on-line scheduling algorithms. *Information Processing Letters*, 51(5):219–222, 1994.

[28] B. Chen, A. van Vliet, and G. J. Woeginger. An optimal algorithm for preemptive on-line scheduling. In J. van Leeuwen, editor, *Algorithms — ESA '94*, pages 300–306, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[29] T. Cheng, H. Kellerer, and V. Kotov. Algorithms better than lpt for semi-online scheduling with decreasing processing times. *Operations Research Letters*, 40(5):349–352, 2012.

[30] M. C. Chou, H. Liu, M. Queyranne, and D. Simchi-Levi. On the asymptotic optimality of a simple on-line algorithm for the stochastic single-machine weighted completion time problem and its extensions. *Operations Research*, 54(3):464–474, 2006.

[31] G. B. Dantzig. Linear programming under uncertainty. *Management Science*, 1(3-4):197–206, 1955.

[32] J. Dohrau. Online makespan scheduling with sublinear advice. In G. F. Italiano, T. Margaria-Steffen, J. Pokorný, J.-J. Quisquater, and R. Wattenhofer, editors, *SOFSEM 2015: Theory and Practice of Computer Science*, pages 177–188, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[33] F. Dufossé, C. Dürr, N. Nadal, D. Trystram, and Óscar C. Vásquez. Scheduling with a processing time oracle. arXiv preprint, 2021.

[34] C. Dürr, T. Erlebach, N. Megow, and J. Meißner. An adversarial model for scheduling with testing. *Algorithmica*, 82(12):3630–3675, 2020.

[35] 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 2018)*, volume 94 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[36] M. Englert, D. Özmen, and M. Westermann. The power of reordering for online minimum makespan scheduling. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 603–612, 2008.

[37] L. Epstein. A survey on makespan minimization in semi-online environments. *Journal of Scheduling*, 21(3):269–284, 2018.

[38] T. Erlebach and M. Hoffmann. Query-competitive algorithms for computing with uncertainty. *Bulletin of EATCS*, 2(116), 2015.

[39] T. Erlebach, M. Hoffmann, and M. S. de Lima. Round-Competitive Algorithms for Uncertainty Problems with Parallel Queries. In M. Bläser and B. Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*, volume 187 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[40] T. Erlebach, M. Hoffmann, M. S. de Lima, N. Megow, and J. Schlöter. Untrusted predictions improve trustable query policies. arXiv preprint, 2021.

[41] T. Erlebach, M. Hoffmann, and F. Kammer. Query-competitive algorithms for cheapest set problems under uncertainty. *Theoretical Computer Science*, 613:51–64, 2016.

[42] T. Erlebach, M. Hoffmann, D. Krizanc, M. Mihal'ák, 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*, volume 1 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 277–288, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[43] U. Faigle, W. Kern, and G. Turan. On the performance of on-line algorithms for partition problems. *Acta cybernetica*, 9(2):107–119, 1989.

[44] 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.

[45] 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.

[46] R. Fleischer and M. Wahl. On-line scheduling revisited. *Journal of Scheduling*, 3(6):343–353, 2000.

[47] G. Galambos and G. J. Woeginger. An on-line scheduling heuristic with better worst-case ratio than graham's list scheduling. *SIAM Journal on Computing*, 22(2):349–355, 1993.

[48] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[49] J. Gittins, K. Glazebrook, and R. Weber. *Multi-armed bandit allocation indices*. John Wiley & Sons, 2 edition, 2011.

[50] M. Goerigk, M. Gupta, J. Ide, A. Schöbel, and S. Sen. The robust knapsack problem with queries. *Computers and Operations Research*, 55:12–22, 2015.

[51] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966.

[52] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[53] A. Gupta, H. Jiang, Z. Scully, and S. Singla. The markovian price of information. In A. Lodi and V. Nagarajan, editors, *Integer Programming and Combinatorial Optimization*, pages 233–246, Cham, 2019. Springer International Publishing.

[54] M. Gupta, Y. Sabharwal, and S. Sen. The update complexity of selection and related problems. In S. Chakraborty and A. Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011)*, volume 13 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 325–338, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[55] M. M. Halldórsson and M. S. de Lima. Query-competitive sorting with uncertainty. *Theoretical Computer Science*, 867:50–67, 2021.

[56] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *J. ACM*, 34(1):144–162, Jan. 1987.

[57] J. A. Hoogeveen and A. P. A. Vestjens. Optimal on-line algorithms for single-machine scheduling. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Integer Programming and Combinatorial Optimization*, pages 404–414, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[58] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954.

[59] S. Kahan. A model for data in motion. In *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*, STOC '91, pages 265–277, New York, NY, USA, 1991. ACM.

[60] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, July 2000.

[61] D. R. Karger, S. J. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20(2):400–430, 1996.

[62] A. J. Keith and D. K. Ahner. A survey of decision making and optimization under uncertainty. *Annals of Operations Research*, 300(2):319–353, 2021.

[63] H. Kellerer, V. Kotov, and M. Gabay. An efficient algorithm for semi-online multiprocessor scheduling with given total processing time. *Journal of Scheduling*, 18(6):623–630, 2015.

[64] H. Kellerer, V. Kotov, M. G. Speranza, and Z. Tuza. Semi on-line algorithms for the partition problem. *Operations Research Letters*, 21(5):235–242, 1997.

[65] S. Khanna and W.-C. Tan. On computing functions with uncertainty. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '01, pages 171–182, New York, NY, USA, 2001. Association for Computing Machinery.

[66] J. Lenstra, A. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. In P. Hammer, E. Johnson, B. Korte, and G. Nemhauser, editors, *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 343–362. Elsevier, 1977.

[67] R. Levi, T. Magnanti, and Y. Shaposhnik. Scheduling with testing. *Management Science*, 65(2):776–793, 2019.

[68] 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.

[69] N. Megow, M. Uetz, and T. Vredeveld. Models and algorithms for stochastic online scheduling. *Mathematics of Operations Research*, 31(3):513–525, 2006.

[70] A. I. Merino and J. A. Soto. The Minimum Cost Query Problem on Matroids with Uncertainty Areas. In C. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 83:1–83:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[71] 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 2000)*, VLDB '00, pages 144–155, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[72] C. Phillips, C. Stein, and J. Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82(1):199–223, 1998.

[73] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer International Publishing, 5 edition, 2016.

[74] C. N. Potts and V. A. Strusevich. Fifty years of scheduling: a survey of milestones. *Journal of the Operational Research Society*, 60(1):S41–S68, 2009.

[75] K. Pruhs, J. Sgall, and E. Torng. Online scheduling. In J. Y.-T. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press Company, 2004.

[76] J. F. Rudin III. Improved bounds for the on-line scheduling problem. Ph.D. Thesis, 2001.

[77] P. Sanders, N. Sivadasan, and M. Skutella. Online scheduling with bounded migration. *Mathematics of Operations Research*, 34(2):481–498, 2009.

[78] L. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16:687–690, 1968.

[79] S. S. Seiden. Preemptive multiprocessor scheduling with rejection. *Theoretical Computer Science*, 262(1):437–458, 2001.

[80] J. Sgall. A lower bound for randomized on-line multiprocessor scheduling. *Information Processing Letters*, 63(1):51–55, 1997.

[81] S. Singla. The price of information in combinatorial optimization. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 2523–2532, USA, 2018. Society for Industrial and Applied Mathematics.

[82] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[83] W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

[84] A. L. Soyster. Technical note—convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, 21(5):1154–1157, 1973.

[85] M. L. Weitzman. Optimal search for the best alternative. *Econometrica*, 47(3):641–654, 1979.

[86] Wolfram Research, Inc.. Mathematica, Version 12.2. Champaign, Illinois, 2020.

[87] A. C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 222–227, 1977.

[88] G. Zhang. A simple semi on-line algorithm for p2//cmax with a buffer. *Information Processing Letters*, 61(3):145–148, 1997.

Appendix: Published First-Author Papers

A Explorable Uncertainty in Scheduling with Non-uniform Testing Times

S. Albers and A. Eckl. Explorable Uncertainty in Scheduling with Non-uniform Testing Times. In C. Kaklamanis and A. Levin, editors, *Approximation and Online Algorithms*, Lecture Notes in Computer Science, pages 127–142, Cham, 2021. Springer International Publishing. [6]

Summary. We study the scheduling with testing setting on a single machine extended to non-negative testing times, and consider the objectives of minimizing the makespan and the sum of completion times. Thereby we extend the original setting of Dürr et al. [35]. All jobs have unknown processing times which can be explored by preliminary tests. Jobs can be run in one of two configurations: they can be executed untested for the duration of a given upper limit, or they are tested and then at some later point executed for the duration of the revealed processing time.

For the makespan objective, we provide optimal algorithms and lower bounds. The deterministic bound is $\varphi \approx 1.6180$ and the randomized bound is $4/3$. Hence, we directly extend the results of Dürr et al. [35] who provided the same ratios for the case when all testing times are equal to 1.

For the sum of completion times objective, we present competitive ratios of 4 and $2\varphi \approx 3.2361$ in the non-preemptive and preemptive settings, respectively. In the preemptive setting, we also provide a lower bound of 1.8546. In the randomized setting, we present an algorithm that is 3.3794-competitive.

Author's Contributions. Development and refinement of the generalized approach for scheduling with testing. Conception, development, and analysis of all new algorithms, lower bounds, and corresponding proofs in the paper. Composition and review of relevant literature. Writing and revision of the paper text, including draft versions and final version. Creation of the included *Mathematica* code used in the analysis for the randomized algorithm.

Full Version. The full version, containing all detailed proofs and concepts, can be found online under https://arxiv.org/abs/2009.13316.

S. Albers and A. Eckl. Explorable Uncertainty in Scheduling with Non-uniform Testing Times. arXiv preprint, 2021. [5]

Explorable Uncertainty in Scheduling with Non-uniform Testing Times

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

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

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

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

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

1 Introduction

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

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

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

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

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

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

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

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

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

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

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

1.1 Problem Statement

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

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

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

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

1.2 Related Work

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

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

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

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

1.3 Contribution

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

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

Table 1. Overview of results

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

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

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

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

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

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

2 Deterministic Setting

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

2.1 Basic Algorithm and Proof of 4-Competitiveness

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

Algorithm 1: (α, β)-SORT

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

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

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

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

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

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

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

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

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

Additionally, let

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Finally, we sum over all jobs j:

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

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

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

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

2.2 A Deterministic Algorithm with Preemption

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

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

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

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

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

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

Algorithm 2: Golden Round Robin

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

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

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

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

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

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

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

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

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

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

3 Randomized Setting

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

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

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

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

Algorithm 3: Randomized-SORT

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

where we define

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

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

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

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

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

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

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

To conclude the proof of the theorem, we write

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

4 Optimal Results for Minimizing the Makespan

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

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

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

5 Conclusion

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

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

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

References

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

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

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

B Scheduling with Testing on Multiple Identical Parallel Machines

The following paper has been published as a peer-reviewed conference paper. The final author's manuscript is included in this chapter. The original published version can be found under https://doi.org/10.1007/978-3-030-83508-8_3.

> Final author's manuscript of: S. Albers and A. Eckl. Scheduling with Testing on Multiple Identical Parallel Machines. In A. Lubiw and M. Salavatipour, editors, *Algorithms and Data Structures*, Lecture Notes in Computer Science, pages 29–42, Cham, 2021. Springer International Publishing. [7]

Summary. We investigate the scheduling with testing setting on multiple identical parallel machines with non-negative testing times, and consider the objective of makespan minimization. The setting was originally introduced by Dürr et al. [35]. Jobs have an unknown processing time which can be revealed by running a preliminary test. The algorithm must assign jobs to one or more machines of the instance, depending on the number of times jobs can be interrupted. Jobs are assigned in one of two possible configurations: they are either run untested, for which the needed time is given by an upper limit, or they are tested and then executed, which takes a total time equal to sum of the given testing time and the revealed processing time.

For the non-preemptive setting, where jobs cannot be interrupted at all, we provide an algorithm whose competitive ratio is 3.1016. This result is compared to a greedy strategy with ratio $2\varphi \approx 3.2361$, and a lower bound of value 2. In the case of uniform testing times, the algorithm can be adapted to be 3-competitive.

For the so-called test-preemptive case, we prove a 2-competitive algorithm. For the normal preemptive case, we provide a lower bound which approaches the value 2 if the number of machines becomes large. In this case, the given results in the test-preemptive and preemptive settings are tight.

Finally, we introduce the so-called *fully-online* setting where jobs arrive one by one, for which we present improved lower bounds.

Author's Contributions. Development and refinement of the scheduling with testing setting for multiple machines. Conception, development, and analysis of all new algorithms, lower bounds, and corresponding proofs in the paper. Composition and review of relevant literature. Writing and revision of the paper text, including draft versions and final version.

Full Version. The full version, containing all detailed proofs and concepts, can be found online under https://arxiv.org/abs/2105.02052.

> S. Albers and A. Eckl. Scheduling with Testing on Multiple Identical Parallel Machines. arXiv preprint, 2021. [8]

Scheduling with Testing on Multiple Identical Parallel Machines*

Susanne Albers[1] and Alexander Eckl[1,2,**]

[1] Department of Informatics, Technical University of Munich,
Boltzmannstr. 3, 85748 Garching, Germany
albers@in.tum.de, alexander.eckl@tum.de
[2] Advanced Optimization in a Networked Economy, Technical University of Munich,
Arcisstraße 21, 80333 Munich, Germany

Abstract. Scheduling with testing is a recent online problem within the framework of explorable uncertainty motivated by environments where some preliminary action can influence the duration of a task. Jobs have an unknown processing time that can be explored by running a test. Alternatively, jobs can be executed for the duration of a given upper limit. We consider this problem within the setting of multiple identical parallel machines and present competitive deterministic algorithms and lower bounds for the objective of minimizing the makespan of the schedule. In the non-preemptive setting, we present the SBS algorithm whose competitive ratio approaches 3.1016 if the number of machines becomes large. We compare this result with a simple greedy strategy and a lower bound which approaches 2. In the case of uniform testing times, we can improve the SBS algorithm to be 3-competitive. For the preemptive case we provide a 2-competitive algorithm and a tight lower bound which approaches the same value.

Keywords: Online Scheduling · Identical Parallel Machines · Explorable Uncertainty · Makespan Minimization · Competitive Analysis

1 Introduction

One of the most fundamental problems in online scheduling is makespan minimization on multiple parallel machines. An online sequence of n jobs with processing times p_j has to be assigned to m identical machines. The objective is to minimize the makespan of the schedule, i.e. the maximum load on any machine. In 1966, Graham [25] showed that the List Scheduling algorithm, which assigns every job to the currently least loaded machine, is $(2 - \frac{1}{m})$-competitive. Since then the upper bound has been improved multiple times, most recently to 1.9201

* Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 277991500/GRK2201, and by the European Research Council, Grant Agreement No. 691672.

** Corresponding author, eMail: alexander.eckl@tum.de

by Fleischer and Wahl [22]. At the same time, the lower bound has also been the focus of a lot of research, the current best result is 1.88 by Rudin [37].

We consider this classical problem in the framework of *explorable uncertainty*, where part the input is initially unknown to the algorithm and can be explored by investing resources which are added as costs to the objective function. Let n jobs be given. Every job j has a processing time p_j and an upper bound u_j. It holds $0 \leq p_j \leq u_j$ for all j. Each job also has a testing time $t_j \geq 0$. A job can be executed on one of m identical machines in one of two modes: It can either be run untested, which takes time u_j, or be tested and then executed, which takes a total time of $t_j + p_j$. The number of jobs n, as well as all testing times t_j and upper bounds u_j are known to the algorithm in the beginning. In particular, an algorithm can sort/order the jobs in a convenient way based on these parameters. The processing time p_j for job j is revealed once the test t_j is completed. This *scheduling with testing* setting has been recently studied by Dürr et al. [13], and Albers and Eckl [3] on a single machine.

We differentiate between *preemptive* and *non-preemptive* settings: If preemption is allowed, a job may be interrupted at any time, and then continued later on a possibly different machine. No two machines may work on the same job at the same time. In case a job is tested, any section of the test must be scheduled earlier than any section of the actual job processing. In the non-preemptive setting, a job assigned to a machine has be fully scheduled without interruption on this machine, independent of whether it is tested or not. We also introduce the notion of *test-preemptive* scheduling, where a job can only be interrupted right after its test is completed.

Scheduling with testing is well-motivated by real world settings where a preliminary evaluation or operation can be executed to improve the duration or difficulty of a task. Examples for the case of multiple machines include e.g. a manufacturing plan where a number of jobs with uncertain length have to be assigned to multiple workers, or a distributed computing setting where tasks with unknown parameters have to be allocated to remote computing nodes by a central scheduler. Several examples for applicable settings for scheduling with testing can also be found in [3,13].

In summary, we study the classical problem of makespan minimization on identical parallel machines in the framework of explorable uncertainty. We use competitive analysis to compare the value of an algorithm with an optimal offline solution. The setting closely relates to online machine scheduling problems studied previously in the literature. We investigate deterministic algorithms and lower bounds for the preemptive and non-preemptive variations of this problem.

1.1 Related Work

Scheduling with testing describes the setting where jobs with uncertain processing times have to be scheduled tested or untested on a given number of machines. The problem has been first studied by Dürr et al. [13,14] for the special case of scheduling jobs on a single machine with uniform testing times $t_j \equiv 1$. They presented several algorithms and lower bounds for the objectives of the sum of

completion times and the makespan. More recently, Albers and Eckl [3] considered the one machine case with testing times $t_j \in \mathbb{N}$, presenting generalized algorithms for both objectives. In this paper, we consider scheduling with testing on identical parallel machines, a natural generalization of the previously studied one machine case.

Makespan minimization in online scheduling with identical machines has been studied extensively in the past decades, ever since Graham [25] established his $(2 - \frac{1}{m})$-competitive List Scheduling algorithm in 1966. In the deterministic setting, a series of publications improved Graham's result to competitive ratios of $2 - \frac{1}{m} - \varepsilon_m$ [23] where $\varepsilon_m \to 0$ for large m, 1.985 [8], 1.945 [31], and 1.923 [1], before Fleischer and Wahl [22] presented the current best result of 1.9201. In terms of the deterministic lower bound for general m, research has been just as fruitful. The bound was improved from 1.707 [19], to 1.837 [9], and 1.852 [1]. The best currently known bound of 1.88 is due to Rudin [37]. For the randomized variant, the lower bound has a current value of $\frac{e}{e-1} \approx 1.582$ [11, 38], while the upper bound is 1.916 [2]. For the deterministic preemptive setting, Chen et al. [12] provide a tight bound of $\frac{e}{e-1}$ for large values of m.

More recently, various extension of this basic case have emerged. In *resource augmentation* settings the algorithm receives some extra resources like machines with higher speed [30], parallel schedules [6, 33], or a reordering buffer [15, 33]. A variation that is closely related to our setting is *semi-online scheduling*, where some additional piece of information is available to the online algorithm in advance. Possible pieces of information include for example the sum of all processing times [5, 32, 33], the value of the optimum [7], or information about the job order [26]. Refer also to the survey by Epstein [16] for an overview of makespan minimization in semi-online scheduling.

Scheduling with testing is directly related to *explorable uncertainty*, a research area concerned with obtaining additional information of unknown parameters through queries with a given cost. Kahan [29] pioneered this line of research in 1991 by studying approximation guarantees for the number of queries necessary to obtain the maximum and median value of a set of uncertain elements. Following this, a variety of problems have been studied in this setting, for example finding the median or k-smallest value [21, 27, 34], geometric tasks [10], caching [36], as well as combinatorial problems like minimum spanning tree [18, 35], shortest path [20], and knapsack [24]. We refer to the survey by Erlebach and Hoffmann [17] for an overview. In the scheduling with testing model, the cost of the queries is added to the objective function. Similar settings are considered for example in Weitzman's pandora's box problem [40], or in the recent 'price of information' model by Singla [39].

1.2 Contribution

In this paper we provide the first results for makespan minimization on multiple machines with testing. We differentiate between general tests $t_j \in \mathbb{N}$ and uniform tests $t_j = 1$, and consider non-preemptive as well as preemptive environments. In

Table 1, we illustrate our results for these cases. The parameter m corresponds to the number of machines in the instance.

Table 1. Overview of results

Setting	General tests	Uniform tests	Lower bound
Non-preemptive	$c(m) \xrightarrow[m\to\infty]{} 3.1016$	$c_1(m) \xrightarrow[m\to\infty]{} 3$	$\max(\varphi, 2-\frac{1}{m})$
Preemptive	2	2	$\max(\varphi, 2-\frac{2}{m}+\frac{1}{m^2})$

In the non-preemptive setting, we present our main algorithm with competitive ratio $c(m)$, which we refer to as the *SBS algorithm*. The function $c(m)$ is increasing in m and has a value of approximately 3.1016 for $m \to \infty$. For uniform tests, we can improve the algorithm to a competitive ratio of $c_1(m)$, which approaches 3 for large values of m. Additionally, we analyze a simple Greedy algorithm for general tests with a competitive ratio of $\varphi(2-\frac{1}{m})$, where $\varphi \approx 1.6180$ is the golden ratio. We also provide a lower bound with value $\max(\varphi, 2-\frac{1}{m})$. The values of $c(m)$, $c_1(m)$, the Greedy algorithm and the lower bound are summarized in Table 2. For all values of $m > 1$ the SBS algorithm has better ratios compared to Greedy. At the same time, the uniform version of the algorithm improves these results further. Though our algorithms work for any number of machines m, they all achieve the same ratio for $m = 1$ as was already proven in [13] and [3] for uniform and general tests, respectively.

If the scheduler is allowed to use preemption, we obtain a 2-approximation for both general and uniform tests. The result holds even in the more restrictive test-preemptive setting. The corresponding lower bound of $\max(\varphi, 2-\frac{2}{m}+\frac{1}{m^2})$ is tight when the number of machines becomes large.

We utilize various methods for our algorithms and lower bounds. The Greedy algorithm we present is a variation of the well-known List Scheduling algorithm introduced by Graham [25]. For the more involved SBS algorithm and its uniform version we employ testing rules for jobs based on the ratio between their upper bound and testing time similar to [3]. We additionally divide the schedule into phases based on these ratios, therefore sorting the jobs by the given parameters to guarantee competitiveness. In the preemptive setting, we divide the schedule into two independent phases, testing and execution, and use an offline algorithm for makespan minimization to solve each instance separately. Lastly, the lower

Table 2. Results in the non-preemptive setting for selected values of m

	1	**2**	**3**	**4**	**5**	**10**	**100**	∞
Greedy	1.6180	2.4271	2.6967	2.8316	2.9125	3.0743	3.2199	3.2361
SBS	1.6180	2.3806	2.6235	2.7439	2.8158	2.9591	3.0874	3.1016
Uniform-SBS	1.6180	2.3112	2.5412	2.6560	2.7248	2.8625	2.9862	3
Lower Bound	1.6180	1.6180	1.6667	1.75	1.8	1.9	1.99	2

bounds we provide are loosely based on a common construction for the classical makespan minimization setting on multiple machines, where a large number of small jobs is followed by a single larger job.

The rest of the paper is structured in the following way: We start by giving some general definitions needed for later sections. In Section 2 we then first prove the competitive ratio of Greedy and the lower bound, before describing the main algorithm for the general case. At the end of the section, we then build a special version of the algorithm for the uniform case. In Section 3 we consider the preemptive setting and give an algorithm as well as a tight lower bound. We conclude the paper by describing some open problems.

1.3 Preliminary Definitions

We use the following notations throughout the document: For a job $j \in [n]$, the *optimal offline running time* of j, i.e. the time needed by the optimum to schedule j on a machine, is denoted as $\rho_j := \min(t_j + p_j, u_j)$, while the *algorithmic running time* of j, i.e. the time needed for an algorithm to run j on a machine, is given by

$$p_j^A := \begin{cases} t_j + p_j & \text{if } j \text{ is tested,} \\ u_j & \text{if } j \text{ is not tested.} \end{cases} \tag{1}$$

It is clear that $\rho_j \leq p_j^A$ for any job j. Additionally, it holds that $p_j \leq \rho_j$, since the processing times p_j are upper bounded by u_j.

At times, we may use the definition of the *minimal running time* of job j, which is given by $\tau_j := \min(t_j, u_j)$.

It is clear that any job must fulfill $\tau_j \leq \rho_j$. In total, we get the following estimation for the different running times:

$$\tau_j \leq \rho_j \leq p_j^A, \qquad \forall j \in [n] \tag{2}$$

Since an algorithm does not know the values p_j, the testing decisions for the jobs are non-trivial. A partial goal for any competitive algorithm is to define a testing scheme such that the algorithmic running times are not too large compared to the optimal offline running times. We provide the following result which was used previously in [3] and is based on Theorem 14 of [13]. The given testing scheme based on the ratio $r_j = u_j/t_j$ between upper bound and testing time is used multiple times within this paper.

Proposition 1. *Let job j be tested iff $r_j \geq \alpha$ for some $\alpha \geq 1$. Then:*

(a) $\forall j \in [n]$ *tested:* $p_j^A \leq \left(1 + \frac{1}{\alpha}\right) \rho_j$
(b) $\forall j \in [n]$ *not tested:* $p_j^A \leq \alpha \rho_j$

As a direct consequence of Proposition 1, an optimal testing scheme for a single job is given by setting the threshold α to the golden ratio $\varphi \approx 1.6180$ [13].

2 Non-preemptive Setting

In this section we assume that preemption is not allowed. Any job has to be assigned to one of m available machines. Since we only consider makespan minimization, we may assume that there is no idle time on the machines and the actual ordering of the executions on a machine does not influence the outcome of the objective. It is therefore sufficient to only consider the assignment of the jobs to the machines.

2.1 Lower Bound and Greedy Algorithm

We first prove a straightforward lower bound and extend the simple List Scheduling algorithm from the classical setting to our problem.

For the lower bound we choose negligibly small testing times coupled with very large upper bounds. This forces the algorithm to test all jobs and thus having to decide on a machine for a given job while having no information about its real execution time.

Theorem 1. *No online algorithm is better than* $(2 - \frac{1}{m})$*-competitive for the problem of makespan minimization on* m *identical machines with testing, even if all testing times are equal to* 1*.*

We note that $\varphi \approx 1.6180$ is always a lower bound for our problem (see [13]), which is relevant only for small values of $m \leq 2$. The proof of Theorem 1 is provided in the full version of this paper [4].

To prove a simple upper bound, we can generalize the List Scheduling algorithm to our problem variant as follows:

Consider the given jobs in any order. For a job j to be scheduled next, test j if and only if $u_j/t_j \geq \varphi$ and then execute it completely on the current least-loaded machine.

Theorem 2. *The extension of List Scheduling described above is* $\varphi(2 - \frac{1}{m})$*-competitive for minimizing the makespan on* m *identical machines with non-uniform testing, where* $\varphi \approx 1.6180$ *is the golden ratio. This analysis is tight.*

The proof structure is similar to the proof of List Scheduling and uses common lower bounds for makespan minimization. We again refer to the full version for all details.

2.2 SBS Algorithm

In this section we provide a 3.1016-competitive algorithm for the non-preemptive setting. It assigns jobs into three classes S_1, B, and S_2 based on their ratios between upper bounds and testing times.

Let $[n]$ be the set of all jobs. We define a threshold function $T(m)$ for all m and divide the jobs into disjoint sets $[n] = B \dot\cup S$, where S will be further subdivided into S_1 and S_2. The set B corresponds to jobs where the ratio $r_j = u_j/t_j$ between

upper bound and testing time is large, while jobs in S have a small ratio. We define

$$B := \{j \in [n] : r_j \geq T(m)\},$$
$$S := [n] \setminus B.$$

For the set S, we would like the algorithm to be able to distinguish jobs based on their optimal offline running time ρ_j. Of course, without testing the algorithm does not know these values, so we instead use the minimal running time τ_j, which can be computed directly using offline input only, to divide the set S further.

We define $S_1 \subset S$, such that $|S_1| = \min(m, |S|)$ and $\forall j_1 \in S_1, j_2 \in S \setminus S_1$: $\tau_{j_1} \geq \tau_{j_2}$. In other words, S_1 is the set of at most m jobs in S with the largest minimal running times. If this definition of S_1 is not unique, we may choose any such set. We set $S_2 := S \setminus S_1$. It follows that if $|S| \leq m$, then $S_2 = \emptyset$.

The idea behind dividing S into two sets is to identify the m largest jobs according to minimal running time and schedule them first, each on a separate machine. This allows us to lower bound the runtime of the remaining jobs later in the schedule.

In Algorithm 1 we describe the SBS algorithm which solves the non-uniform case and works in three phases corresponding to the sets S_1, B and S_2:

Algorithm 1: SBS algorithm

```
B ← {j ∈ [n] : r_j ≥ T(m)};
S ← [n] \ B;
S_1 ← S' ⊂ S s.t. |S'| = min(m, |S|), τ_{j_1} ≥ τ_{j_2} ∀j_1∈S', j_2∈S\S';
S_2 ← S \ S_1;
foreach j ∈ S_1 do
    if r_j ≥ φ then
        test and run j on an empty machine;
    else
        run j untested on an empty machine;
    end
end
foreach j ∈ B do
    test and run j on the current least-loaded machine;
end
foreach j ∈ S_2 do
    run j untested on the current least-loaded machine;
end
```

In order to have a non-trivial testing decision for jobs in S_1, it makes sense to require that $T(m) \geq \varphi$ for all m. More specifically, we will define the threshold function $T(m)$ in the non-uniform setting as follows:

$$T(m) = \frac{(3+\sqrt{5})m - 2 + \sqrt{(38+6\sqrt{5})m^2 - 4(11+\sqrt{5})m + 12}}{6m-2}$$

Theorem 3. *Let $T(m)$ be a parameter function of m defined as above. The SBS algorithm is $T(m)\left(\frac{3}{2}-\frac{1}{2m}\right)$-competitive for minimizing the makespan on m identical machines with non-uniform testing.*

The function $T(m)$ is increasing for all $m \geq 1$ and fulfills $T(1) = \varphi$ as well as approximately $T(m) \to 2.0678$ for $m \to \infty$. The competitive ratio of the algorithm is explicitly given by

$$c(m) = \frac{(3+\sqrt{5})m - 2 + \sqrt{(38+6\sqrt{5})m^2 - 4(11+\sqrt{5})m + 12}}{4m}.$$

For this function we have $c(1) = \varphi$ as well as approximately $c(m) \to 3.1016$ if m approaches infinity. Additionally, it holds that $c(m) < \varphi\left(2-\frac{1}{m}\right)$ for all $m > 1$.

Proof. We assume w.l.o.g. that the job indices are sorted by non-increasing optimal offline running times $\rho_1 \geq \cdots \geq \rho_n$. We denote the last job to finish in the schedule of the algorithm as l and the minimum machine load before job l as t. It follows that the value of the algorithm is $t + p_l^A$.

The value of the optimum is at least as large as the average sum of the optimal offline running times, or

$$L := \frac{1}{m}\sum_{j\in[n]} \rho_j \leq \mathrm{OPT}, \tag{3}$$

since in any schedule at least one machine must have a load of at least this average. At the same time, we know that the optimum has to schedule every job on some machine:

$$\rho_j \leq \mathrm{OPT} \quad \forall j \in [n] \tag{4}$$

We also utilize another common lower bound in makespan minimization, which is the sum of the processing times of the m-th and $(m+1)$-th largest job. If there are at least $m+1$ jobs, then some machine has to schedule at least 2 of these jobs:

$$\rho_m + \rho_{m+1} \leq \mathrm{OPT}\,. \tag{5}$$

Here, ρ_j is defined as 0 if the instance has less than j jobs.

We differentiate between jobs handled by the algorithm in different phases and bound the algorithmic running times against the optimal offline running times. We write $p_j^A \leq \alpha_j \rho_j$ and define different values for α_j depending on the set j belongs to. It holds that

$$\alpha_j = \begin{cases} \varphi & \text{if } j \in S_1, \\ 1+\frac{1}{T(m)} & \text{if } j \in B, \\ T(m) & \text{if } j \in S_2, \end{cases} \tag{6}$$

by Proposition 1 and the testing strategy of the algorithm.

The objective value of the algorithm depends on the set job l belongs to, so we differentiate between three cases. The following proposition upper bounds the algorithmic value $\mathrm{ALG} = t + p_l^A$ for each of these cases:

Proposition 2. *The value of the algorithm can be estimated as follows:*

$$\mathrm{ALG} \leq \begin{cases} \varphi\,\mathrm{OPT} & \text{if } l \in S_1, \\ \left(\varphi + \left(1 + \frac{1}{T(m)}\right)\left(1 - \frac{1}{m}\right)\right)\mathrm{OPT} & \text{if } l \in B, \\ T(m)\left(\frac{3}{2} - \frac{1}{2m}\right)\mathrm{OPT} & \text{if } l \in S_2. \end{cases}$$

To prove this proposition, we utilize the lower bounds (3)-(5) and the estimates (6) for the value of α_j. A critical step lies in the estimation of p_l^A for $l \in S_2$, where we are able to lower bound τ_l using the size of the m-th and $(m{+}1)$-th largest job because the algorithm already ran m jobs from S_1 in the beginning of the schedule. We refer to the full version [4] for a detailed proof.

It remains to take the maximum over all three cases and minimize the value in dependence of $T(m)$. The value in the case $l \in S_1$ is always less than the values given by the other cases, therefore we only want to minimize

$$\max\left(\varphi + \left(1 + \frac{1}{T(m)}\right)\left(1 - \frac{1}{m}\right),\ T(m)\left(\frac{3}{2} - \frac{1}{2m}\right)\right).$$

The left side of the maximum is decreasing in $T(m)$, while the right side is increasing. The minimal maximum is therefore attained when both sides are equal. It can be easily verified that for the given definition of the threshold function $T(m)$ both sides of the maximum are equal for all values of $m \geq 1$.

It follows that the final ratio can be estimated by $\frac{\mathrm{ALG}}{\mathrm{OPT}} \leq T(m)\left(\frac{3}{2} - \frac{1}{2m}\right)$. □

2.3 An Improved Algorithm for the Uniform Case

The previous section established an algorithm with a competitive ratio of approximately 3.1016. We now present an algorithm with a better ratio in the case when $t_j = 1$ for all jobs. We define the threshold function $T_1(m)$ as follows:

$$T_1(m) = \frac{2m - 1 + \sqrt{16m^2 - 14m + 3}}{3m - 1}$$

The *Uniform-SBS* algorithm works as follows: Sort the jobs by non-increasing u_j. Go through the sorted list of jobs and put the next job on the machine with the lowest current load. A job j is tested if $u_j \geq T_1(m)$, otherwise it is run untested.

Theorem 4. *Uniform-SBS is a* $T_1(m)(\frac{3}{2} - \frac{1}{2m})$*-competitive algorithm for uniform instances.*

For uniform jobs with $t_j = 1$, sorting by non-increasing upper bound u_j is consistent with sorting by non-increasing ratio r_j. Hence, Uniform-SBS is similar to the SBS algorithm reduced to the phases corresponding to the sets B and S, where S contains *all* small jobs. The reason behind running the m largest jobs of S first in the SBS algorithm was to upper bound the remaining jobs in S. For uniform testing times, this bound can be achieved *without* this special structure.

The function $T_1(m)$ is increasing for all $m \geq 1$ and fulfills $T_1(1) = \varphi$ as well as $T_1(m) \to 2$ for $m \to \infty$. Computing the competitive ratio explicitly yields

$$c_1(m) = \frac{2m - 1 + \sqrt{16m^2 - 14m + 3}}{2m}.$$

These values start from $c_1(1) = \varphi$ and approach $c_1(m) \to 3$ if $m \to \infty$. Additionally, it holds that $c_1(m) < c(m)$ for all $m > 1$. In other words, this special version of the algorithm is strictly better than the general SBS algorithm described in Section 2.2. We defer the proof of Theorem 4 to the full version of the paper [4].

3 Results with Preemption

In this section we assume that jobs can be preempted at any time during their execution. An interrupted job may be continued on a possibly different machine, but no two machines may work on the same job at the same time. Testing a job must be completely finished before any part of its execution can take place.

It makes sense to additionally consider the following stricter definition of preemption within scheduling with testing: Untested jobs must be run without interruption on a single machine. If a job is tested, its test must also be run without interruption on one machine. The execution after the test may then be run without interruption on a possibly different machine. We call this setting *test-preemptive*, referring to the fact that the only place where we might preempt a job is exactly when its test is completed. From an application point of view, the test-preemptive setting is a natural extension of the non-preemptive setting, allowing the scheduler to reconsider the assignment of a job after receiving more information through the test.

Clearly, the difficulty of settings within scheduling with testing increases in the following order: preemptive, test-preemptive and non-preemptive. We now present the 2-competitive *Two Phases* algorithm for the test-preemptive setting, which can be applied directly to the ordinary preemptive case. Additionally, we construct a lower bound of $2 - 2/m + 1/m^2$ for the ordinary preemptive case. This lower bound then also holds for test-preemption, and is therefore tight for both settings when the number of machines m approaches infinity.

The Two Phases algorithm for the test-preemptive setting works as follows: Let OFF denote an optimal offline algorithm for makespan minimization on m machines. In the first phase, the algorithm schedules all jobs for their minimal running time τ_j using the algorithm OFF. Herein, the algorithm tests all jobs except trivial jobs with $t_j > u_j$, where running the upper bound is optimal. In the second phase, all remaining jobs are already tested, hence the algorithm now knows all remaining processing times p_j. We then use the offline algorithm OFF again to schedule the remaining jobs optimally. Finally, the algorithm obliviously puts the second schedule on top of the first.

Theorem 5. *The Two Phases algorithm is 2-competitive for minimizing the makespan on m machines with testing in the test-preemptive setting.*

The proof makes use of the assumption that the algorithm has access to unlimited computational power, which is a common assumption in online optimization. If we do not give the online algorithm this power, the result is slightly worse, since offline makespan minimization is strongly NP-hard. We may then make use of the PTAS for offline makespan minimization by Hochbaum and Shmoys [28] to achieve a ratio of $2+\varepsilon$ for any $\varepsilon > 0$, where the runtime of the algorithm increases exponentially with $1/\varepsilon$. All details of the proof can be found in the full version [4].

For the lower bound result we now consider the standard preemptive setting where a job can be interrupted at any time.

Theorem 6. *In the preemptive setting, no online algorithm for makespan minimization on m identical machines with testing can have a better competitive ratio than $2 - 2/m + 1/m^2$, even if all testing times are equal to 1.*

We note that $\varphi \approx 1.6180$ also remains a lower bound even for the preemptive case, since two machines cannot run the same job concurrently. It holds $2 - 2/m + 1/m^2 < \varphi$ only for values of $m \leq 4$.

Proof. Let us consider the following example: Let M be a sufficiently large number and let $m(m-1)$ small jobs be given with $t_j = 1, p_j = 0, u_j = M$ as well as one large job f with $t_f = 1, p_f = m-1, u_f = M$. As argued in the proof of Theorem 1, OPT has a value of m and we may assume that the algorithm tests every job.

In the preemptive setting we required that any execution of the actual processing time of a job can only happen after its test is completed, therefore any job j that finished testing at some time t is completed not earlier than $t+p_j$. The adversary decides the processing time of j by the following rule: If $t \geq m-1+1/m$ and job f has not yet been assigned, set $p_j = m-1$ (i.e. set $j = f$). Else, set $p_j = 0$.

If the adversary assigns job f at any point, then job f finished testing at time $t \geq m-1+1/m$. It follows that

$$\text{ALG} \geq t + p_f \geq m - 1 + \frac{1}{m} + m - 1 = 2m - 2 + \frac{1}{m}.$$

Hence the competitive ratio is at least $\frac{\text{ALG}}{\text{OPT}} \geq 2 - \frac{2}{m} + \frac{1}{m^2}$.

All that remains is to show that this assignment of f happens at some point during the runtime of the algorithm. Assume that this is not the case, i.e. all jobs finish testing earlier than $m-1+1/m$. The adversary sets all $p_j = 0$, hence it follows directly that all jobs are completely finished before $m-1+1/m$. But this means that the algorithmic solution has a value of $\text{ALG} < m-1+1/m$.

Since $t_j = 1$ for all jobs, we know that the average load L fulfills

$$L \geq \frac{1}{m}(m(m-1)+1) = m - 1 + 1/m.$$

But L is a lower bound on the optimal value of the instance, even in the preemptive setting, contradicting $\text{ALG} < m-1+1/m$. ◻

4 Conclusion

We presented algorithms and lower bounds for the problem of scheduling with testing on multiple identical parallel machines with the objective of minimizing the makespan. Such settings arise whenever a preliminary action influences cost, duration or difficulty of a task. Our main results were a 3.1016-competitive algorithm for the non-preemptive case and a tight 2-competitive algorithm for the preemptive case if the number of machines becomes large.

Apart from closing the gaps between our ratios and the lower bounds, we propose the following consideration for future work: A natural generalization of our setting is to consider *fully-online* arrivals, where jobs arrive one by one and have to be scheduled immediately. It is clear that this setting is at least as hard as the problem considered in this paper. In the full version [4], we provide a simple lower bound with value 2 for this generalization that holds for all values of $m \geq 2$. An upper bound is clearly given by the Greedy algorithm we provided in Section 2. Finding further algorithms or lower bounds for this new setting is a compelling direction for future research.

References

1. Albers, S.: Better bounds for online scheduling. SIAM Journal on Computing **29**(2), 459–473 (1999). https://doi.org/10.1137/S0097539797324874
2. Albers, S.: On randomized online scheduling. In: Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing. p. 134–143. STOC '02, Association for Computing Machinery, New York, NY, USA (2002). https://doi.org/10.1145/509907.509930
3. Albers, S., Eckl, A.: Explorable uncertainty in scheduling with non-uniform testing times. In: Approximation and Online Algorithms. Lecture Notes in Computer Science, Springer (2021), https://arxiv.org/abs/2009.13316, to appear.
4. Albers, S., Eckl, A.: Scheduling with testing on multiple identical parallel machines (2021), https://arxiv.org/abs/2105.02052
5. Albers, S., Hellwig, M.: Semi-online scheduling revisited. Theoretical Computer Science **443**, 1 – 9 (2012). https://doi.org/10.1016/j.tcs.2012.03.031
6. Albers, S., Hellwig, M.: Online makespan minimization with parallel schedules. Algorithmica **78**(2), 492–520 (2017). https://doi.org/10.1007/s00453-016-0172-5
7. Azar, Y., Regev, O.: On-line bin-stretching. Theoretical Computer Science **268**(1), 17–41 (2001). https://doi.org/10.1016/S0304-3975(00)00258-9
8. Bartal, Y., Fiat, A., Karloff, H., Vohra, R.: New algorithms for an ancient scheduling problem. In: Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing. p. 51–58. STOC '92, Association for Computing Machinery, New York, NY, USA (1992). https://doi.org/10.1145/129712.129718
9. Bartal, Y., Karloff, H., Rabani, Y.: A better lower bound for on-line scheduling. Information Processing Letters **50**(3), 113 – 116 (1994). https://doi.org/10.1016/0020-0190(94)00026-3
10. Bruce, R., Hoffmann, M., Krizanc, D., Raman, R.: Efficient update strategies for geometric computing with uncertainty. Theory of Computing Systems **38**(4), 411–423 (2005). https://doi.org/10.1007/s00224-004-1180-4

11. Chen, B., van Vliet, A., Woeginger, G.J.: A lower bound for randomized on-line scheduling algorithms. Information Processing Letters **51**(5), 219 – 222 (1994). https://doi.org/10.1016/0020-0190(94)00110-3
12. Chen, B., van Vliet, A., Woeginger, G.J.: An optimal algorithm for preemptive on-line scheduling. In: van Leeuwen, J. (ed.) Algorithms — ESA '94. pp. 300–306. Springer Berlin Heidelberg, Berlin, Heidelberg (1994). https://doi.org/10.1007/BFb0049417
13. Dürr, C., Erlebach, T., Megow, N., Meißner, J.: Scheduling with Explorable Uncertainty. In: Karlin, A.R. (ed.) 9th Innovations in Theoretical Computer Science Conference (ITCS 2018). Leibniz International Proceedings in Informatics (LIPIcs), vol. 94, pp. 30:1–30:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2018). https://doi.org/10.4230/LIPIcs.ITCS.2018.30
14. Dürr, C., Erlebach, T., Megow, N., Meißner, J.: An adversarial model for scheduling with testing. Algorithmica **82**(12), 3630–3675 (2020). https://doi.org/10.1007/s00453-020-00742-2
15. Englert, M., Özmen, D., Westermann, M.: The power of reordering for online minimum makespan scheduling. In: 2008 49th Annual IEEE Symposium on Foundations of Computer Science. pp. 603–612 (2008). https://doi.org/10.1109/FOCS.2008.46
16. Epstein, L.: A survey on makespan minimization in semi-online environments. Journal of Scheduling **21**(3), 269–284 (2018). https://doi.org/10.1007/s10951-018-0567-z
17. Erlebach, T., Hoffmann, M.: Query-competitive algorithms for computing with uncertainty. Bulletin of EATCS **2**(116) (2015)
18. Erlebach, T., Hoffmann, M., Krizanc, D., Mihal'ák, M., Raman, R.: Computing Minimum Spanning Trees with Uncertainty. In: Albers, S., Weil, P. (eds.) 25th International Symposium on Theoretical Aspects of Computer Science. Leibniz International Proceedings in Informatics (LIPIcs), vol. 1, pp. 277–288. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2008). https://doi.org/10.4230/LIPIcs.STACS.2008.1358
19. Faigle, U., Kern, W., Turan, G.: On the performance of on-line algorithms for partition problems. Acta cybernetica **9**(2), 107–119 (1989)
20. Feder, T., Motwani, R., O'Callaghan, L., Olston, C., Panigrahy, R.: Computing shortest paths with uncertainty. Journal of Algorithms **62**(1), 1–18 (2007). https://doi.org/10.1016/j.jalgor.2004.07.005
21. Feder, T., Motwani, R., Panigrahy, R., Olston, C., Widom, J.: Computing the median with uncertainty. SIAM Journal on Computing **32**(2), 538–547 (2003). https://doi.org/10.1137/S0097539701395668
22. Fleischer, R., Wahl, M.: On-line scheduling revisited. Journal of Scheduling **3**(6), 343–353 (2000). https://doi.org/10.1002/1099-1425(200011/12)3:6<343::AID-JOS54>3.0.CO;2-2
23. Galambos, G., Woeginger, G.J.: An on-line scheduling heuristic with better worst-case ratio than graham's list scheduling. SIAM Journal on Computing **22**(2), 349–355 (1993). https://doi.org/10.1137/0222026
24. Goerigk, M., Gupta, M., Ide, J., Schöbel, A., Sen, S.: The robust knapsack problem with queries. Computers and Operations Research **55**, 12 – 22 (2015). https://doi.org/10.1016/j.cor.2014.09.010
25. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell System Technical Journal **45**(9), 1563–1581 (1966). https://doi.org/10.1002/j.1538-7305.1966.tb01709.x
26. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics **17**(2), 416–429 (1969). https://doi.org/10.1137/0117039

27. Gupta, M., Sabharwal, Y., Sen, S.: The update complexity of selection and related problems. In: Chakraborty, S., Kumar, A. (eds.) IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011). Leibniz International Proceedings in Informatics (LIPIcs), vol. 13, pp. 325–338. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2011). https://doi.org/10.4230/LIPIcs.FSTTCS.2011.325
28. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems theoretical and practical results. J. ACM **34**(1), 144–162 (Jan 1987). https://doi.org/10.1145/7531.7535
29. Kahan, S.: A model for data in motion. In: Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing. pp. 265–277. STOC '91, ACM, New York, NY, USA (1991). https://doi.org/10.1145/103418.103449
30. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. J. ACM **47**(4), 617–643 (Jul 2000). https://doi.org/10.1145/347476.347479
31. Karger, D.R., Phillips, S.J., Torng, E.: A better algorithm for an ancient scheduling problem. Journal of Algorithms **20**(2), 400 – 430 (1996). https://doi.org/10.1006/jagm.1996.0019
32. Kellerer, H., Kotov, V., Gabay, M.: An efficient algorithm for semi-online multiprocessor scheduling with given total processing time. Journal of Scheduling **18**(6), 623–630 (2015). https://doi.org/10.1007/s10951-015-0430-4
33. Kellerer, H., Kotov, V., Speranza, M.G., Tuza, Z.: Semi on-line algorithms for the partition problem. Operations Research Letters **21**(5), 235 – 242 (1997). https://doi.org/10.1016/S0167-6377(98)00005-4
34. Khanna, S., Tan, W.C.: On computing functions with uncertainty. In: Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. p. 171–182. PODS '01, Association for Computing Machinery, New York, NY, USA (2001). https://doi.org/10.1145/375551.375577
35. Megow, N., Meißner, J., Skutella, M.: Randomization helps computing a minimum spanning tree under uncertainty. SIAM Journal on Computing **46**(4), 1217–1240 (2017). https://doi.org/10.1137/16M1088375
36. Olston, C., Widom, J.: Offering a precision-performance tradeoff for aggregation queries over replicated data. In: 26th International Conference on Very Large Data Bases (VLDB 2000). p. 144–155. VLDB '00, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2000)
37. Rudin III, J.F.: Improved bounds for the on-line scheduling problem. Ph.D. Thesis (2001)
38. Sgall, J.: A lower bound for randomized on-line multiprocessor scheduling. Information Processing Letters **63**(1), 51 – 55 (1997). https://doi.org/10.1016/S0020-0190(97)00093-8
39. Singla, S.: The price of information in combinatorial optimization. In: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms. p. 2523–2532. SODA '18, Society for Industrial and Applied Mathematics, USA (2018). https://doi.org/10.1137/1.9781611975031.161
40. Weitzman, M.L.: Optimal search for the best alternative. Econometrica **47**(3), 641–654 (1979). https://doi.org/10.2307/1910412