Speed-Robust Scheduling
Sand, Bricks, and Rocks

Franziska Eberle[1(✉)], Ruben Hoeksma[2], Nicole Megow[1], Lukas Nölke[1], Kevin Schewior[3], and Bertrand Simon[4]

[1] Faculty of Mathematics and Computer Science, University of Bremen, Bremen, Germany
{feberle,nmegow,noelke}@uni-bremen.de
[2] Department of Applied Mathematics, University of Twente, Enschede, The Netherlands
r.p.hoeksma@utwente.nl
[3] Department of Mathematics and Computer Science, Universität zu Köln, Cologne, Germany
schewior@cs.uni-koeln.de
[4] IN2P3 Computing Center, CNRS, Villeurbanne, France
bertrand.simon@cc.in2p3.fr

Abstract. The speed-robust scheduling problem is a two-stage problem where, given m machines, jobs must be grouped into at most m bags while the processing speeds of the machines are unknown. After the speeds are revealed, the grouped jobs must be assigned to the machines without being separated. To evaluate the performance of algorithms, we determine upper bounds on the worst-case ratio of the algorithm's makespan and the optimal makespan given full information. We refer to this ratio as the robustness factor. We give an algorithm with a robustness factor $2 - \frac{1}{m}$ for the most general setting and improve this to 1.8 for equal-size jobs. For the special case of infinitesimal jobs, we give an algorithm with an optimal robustness factor equal to $\frac{e}{e-1} \approx 1.58$. The particular machine environment in which all machines have either speed 0 or 1 was studied before by Stein and Zhong (SODA 2019). For this setting, we provide an algorithm for scheduling infinitesimal jobs with an optimal robustness factor of $\frac{1+\sqrt{2}}{2} \approx 1.207$. It lays the foundation for an algorithm matching the lower bound of $\frac{4}{3}$ for equal-size jobs.

1 Introduction

Scheduling problems with incomplete knowledge of the input data have been studied extensively. There are different ways to model such uncertainty, the major frameworks being *online optimization*, where parts of the input are revealed incrementally, *stochastic optimization*, where parts of the input are modeled as random variables, and *robust optimization*, where uncertainty in the

Partially supported by the German Science Foundation (DFG) under contract ME 3825/1.

M. Singh and D. P. Williamson (Eds.): IPCO 2021, LNCS 12707, pp. 283–296, 2021.
https://doi.org/10.1007/978-3-030-73879-2_20

data is bounded. Most scheduling research in this context assumes uncertainty about the job characteristics. Examples include online scheduling, where the job set is a priori unknown [1,18], stochastic scheduling, where the processing times are modeled as random variables [17], robust scheduling, where the unknown processing times are within a given interval [14], two/multi-stage stochastic and robust scheduling [5,19], and scheduling with explorable execution times [8,15].

A lot less research addresses uncertainty about the machine environment, particularly, where the processing speeds of machines change in an unforeseeable manner. A majority of such research focuses on the special case of scheduling with unknown non-availability periods, that is, machines break down temporarily [2,7] or permanently [20]. Arbitrarily changing machine speeds have been considered for scheduling on a single machine [10,16].

We consider a two-stage robust scheduling problem with multiple machines of unknown speeds. Given n jobs and m machines, we ask for a partition of the jobs into m groups, we say *bags*, that have to be scheduled on the machines after their speeds are revealed without being split up. That is, in the second stage, when the machine speeds are known, a feasible schedule assigns jobs in the same bag to the same machine. The goal is to minimize the second-stage makespan.

More formally, we define the *speed-robust scheduling* problem as follows. We are given n jobs with processing times $p_j \geq 0$, for $j \in \{1, \ldots, n\}$, and the number of machines, $m \in \mathbb{N}$. Machines run in parallel but their speeds are a priori unknown. In the first stage, the task is to group jobs into at most m bags. In the second stage, the machine speeds $s_i \geq 0$, for $i \in \{1, \ldots, m\}$, are revealed. The time needed to execute job j on machine i is $\frac{p_j}{s_i}$, if $s_i > 0$. If a machine has speed $s_i = 0$, then it cannot process any job; we say the machine *fails*. Given the machine speeds, the second-stage task is to assign the bags to the machines such that the makespan $C_{\max}$ is minimized, where the makespan is the maximum sum of execution times of jobs assigned to the same machine.

Given a set of bags and machine speeds, the second-stage problem emerges as classical makespan minimization on related parallel machines. It is well-known that this problem can be solved arbitrarily close to optimality by polynomial-time approximation schemes [3,12,13]. As we are interested in the information-theoretic tractability and allow superpolynomial running times, ignoring any computational concern, we assume that the second-stage problem is solved optimally. Thus, an *algorithm* for speed-robust scheduling defines a job-to-bag allocation, i.e., it gives a solution to the first-stage problem. We may use non-optimal bag-to-machine allocations to simplify the analysis.

We evaluate the performance of algorithms by a worst-case analysis, comparing an algorithm's makespan with the optimal makespan achievable when machine speeds are known in advance. We say that an algorithm is ρ-*robust* if, for any instance, its makespan is within a factor $\rho \geq 1$ of the optimal solution. The *robustness factor* of the algorithm is defined as the infimum over all such ρ.

The special case of speed-robust scheduling with machine speeds in $\{0, 1\}$ has been studied by Stein and Zhong [20]. They introduced the problem with identical machines and an unknown number of machines that fail (speed 0) in the

second stage. They present a simple lower bound of $\frac{4}{3}$ on the robustness factor with equal jobs and design a general $\frac{5}{3}$-robust algorithm. For infinitesimal jobs, they give a 1.2333-robust algorithm complemented by a lower bound for each number of machines which tends to $\frac{1+\sqrt{2}}{2} \approx 1.207$ for large m. Stein and Zhong also consider the objective of minimizing the maximum difference between the most and least loaded machine, motivated by questions on fair allocation.

Our Contribution

We introduce the speed-robust scheduling problem and present robust algorithms. The algorithmic difficulty of this problem is to construct bags in the first stage that are robust under any choice of machine speeds in the second stage. The straight-forward approach of using any makespan-optimal solution on m identical machines is not sufficient. Lemma 6 shows that such an algorithm might have an arbitrarily large robustness factor. Using *Longest Processing Time* first (LPT) to create bags does the trick and is $\left(2 - \frac{1}{m}\right)$-robust for arbitrary job sizes (Theorem 4). While this was known for speeds in $\{0, 1\}$ [20], our most general result is much less obvious.

Note that LPT aims at "balancing" the bag sizes which cannot lead to a better robustness factor than $2 - \frac{1}{m}$ as we show in Lemma 7. Hence, to improve upon this factor, we need to carefully construct bags with imbalanced bag sizes. There are two major challenges with this approach: (i) finding the ideal imbalance in the bag sizes independent from the actual job processing times that would be robust for all adversarial speed settings simultaneously and (ii) to adapt bag sizes to accommodate discrete jobs.

A major contribution of this paper is an optimal solution to the first challenge by considering infinitesimal jobs (Theorem 1). One can think of this as filling bags with *sand* to the desired level. Thus, the robust scheduling problem boils down to identifying the best bag sizes as placing the jobs into bags becomes trivial. We give, for any number of machines, optimally imbalanced bag sizes and prove a robustness factor of

$$\bar{\rho}(m) = \frac{m^m}{m^m - (m-1)^m} \leq \frac{e}{e-1} \approx 1.58\,.$$

For infinitesimal jobs in the particular machine environment in which all machines have either speed 0 or 1, we obtain an algorithm with robustness factor

$$\bar{\rho}_{01}(m) = \max_{t \leq \frac{m}{2},\ t \in \mathbb{N}} \frac{1}{\frac{t}{m-t} + \frac{m-2t}{m}} \leq \frac{1+\sqrt{2}}{2} \approx 1.207 = \bar{\rho}_{01}\,.$$

This improves the previous upper bound of 1.233 by Stein and Zhong [20] and matches exactly their lower bound for each m. Furthermore, we show that the lower bound in [20] holds even for randomized algorithms and, thus, our algorithm is optimal for both, deterministic and randomized scheduling (Theorem 2).

The above tight results for infinitesimal jobs are crucial for our further results for discrete jobs. Following the figurative notion of sand for infinitesimal jobs,

Table 1. Summary of results on speed-robust scheduling.

	General speeds		Speeds from $\{0,1\}$	
	Lower bound	Upper bound	Lower bound	Upper bound
Discrete jobs (Rocks)	$\bar{\rho}(m)$ (Lemma 1)	$2-\frac{1}{m}$ (Theorem 4)	$\frac{4}{3}$ [20]	$\frac{5}{3}$ [20]
Equal-size jobs (Bricks)	$\bar{\rho}(m)$ (Lemma 1)	1.8 (Theorem 5)	$\frac{4}{3}$ ([20], Theorem 6)	
Infinitesimal jobs (Sand)	$\bar{\rho}(m) \leq \frac{e}{e-1} \approx 1.58$ (Lemma 1, Theorem 1)		$\bar{\rho}_{01}(m) \leq \frac{1+\sqrt{2}}{2} \approx 1.207$ ([20], Theorem 2)	

we think of equal-size jobs as *bricks* and arbitrary jobs as *rocks*. Building on those ideal bag sizes, our approaches differ substantially from the methods in [20]. When all jobs have equal processing time, we obtain a 1.8-robust solution through a careful analysis of the trade-off between using slightly imbalanced bags and a scaled version of the ideal bag sizes computed for the infinitesimal setting (Theorem 5).

When machines have only speeds in $\{0,1\}$ and jobs have arbitrary equal sizes, i.e., unit size, we give an optimal $\frac{4}{3}$-robust algorithm (Theorem 6). This is an interesting class of instances as the best known lower bound of $\frac{4}{3}$ for discrete jobs uses only unit-size jobs [20]. To achieve this result, we exploit the ideal bag sizes computed for infinitesimal jobs by using a scaled variant of these sizes. Some cases, depending on m and the optimal makespan on m machines, have to be handled individually. Here, we use a direct way of constructing bags with at most four different bag sizes and some cases can be solved by an integer linear program. We summarize our results in Table 1.

Inspired by traditional one-stage scheduling problems where jobs have *machine-dependent execution times* (unrelated machine scheduling), one might ask for such a generalization of our problem. However, it is easy to rule out any robustness factor for such a setting: Consider four machines and five jobs, where each job may be executed on a unique pair of machines. Any algorithm must build at least one bag with at least two jobs. For this bag there is at most one machine to which it can be assigned with finite makespan. If this machine fails, the algorithm cannot complete the jobs whereas an optimal solution can split this bag on multiple machines to get a finite makespan.

Due to space constraints, we omit proof details. They can be found in a full version of this paper [9].

2 Speed-Robust Scheduling with Infinitesimal Jobs

In this section, we consider speed-robust scheduling with infinitely many jobs that have infinitesimal processing times. We give optimal algorithms in both the general case and the special case with speeds in $\{0,1\}$.

2.1 General Speeds

Theorem 1. *There is an algorithm for speed-robust scheduling with infinitesimal jobs that is $\bar{\rho}(m)$-robust for all $m \geq 1$, where*

$$\bar{\rho}(m) = \frac{m^m}{m^m - (m-1)^m} \leq \frac{e}{e-1} \approx 1.58\,.$$

This is the best possible robustness factor that can be achieved by any algorithm.

To prove Theorem 1, we first show that, even when we restrict the adversary to a particular set of m speed configurations, no algorithm can achieve a robustness factor better than $\bar{\rho}(m)$. Note that since we can scale all speeds equally by an arbitrary factor without influencing the robustness factor, we can assume that the sum of the speeds equals 1. Similarly, we assume that the total processing time of the jobs equals 1 such that the optimal makespan of the adversary is 1 and the worst-case makespan of an algorithm is equal to its robustness factor.

Intuitively, the idea behind the set of speed configurations is that the adversary can set $m-1$ machines to equal low speeds and one machine to a high speed. The low speeds are set such that one particular bag size just fits on that machine when aiming for the given robustness factor. This immediately implies that all larger bags have to be put on the fast machine together. This way, the speed configuration can *target* a certain bag size. We provide specific bag sizes that achieve a robustness of $\bar{\rho}(m)$ and show that for the speeds targeting these bag sizes, other bag sizes would result in even larger robustness factors.

We define $U = m^m$, $L = m^m - (m-1)^m$, as well as $t_k = (m-1)^{m-k} m^{k-1}$ for $k \in \{1, \ldots, m\}$. Intuitively, these values are chosen such that the bag sizes $\frac{t_i}{L}$ are optimal and $\frac{t_i}{U}$ corresponds to the low speed of the i-th speed configuration. It is easy to verify that $\bar{\rho}(m) = \frac{U}{L}$ and for all k we have

$$\sum_{i<k} t_i = (m-1)t_k - U + L\,. \tag{1}$$

In particular, this implies that $\sum_{i \leq m} t_i = mt_m - U + L = L$, and therefore, the sum of the bag sizes is 1. Let $a_1 \leq \cdots \leq a_m$ denote the bag sizes chosen by an algorithm and $s_1 \leq \cdots \leq s_m$ the speeds chosen by the adversary.

Lemma 1. *For any $m \geq 1$, no algorithm for speed-robust scheduling with infinitesimal jobs can have a robustness factor less than $\bar{\rho}(m)$.*

Proof. We restrict the adversary to the following m speed configurations indexed by $k \in \{1, \ldots, m\}$:

$$\mathcal{S}_k := \left\{ s_1 = \frac{t_k}{U},\ s_2 = \frac{t_k}{U},\ \ldots,\ s_{m-1} = \frac{t_k}{U},\ s_m = 1 - (m-1)\frac{t_k}{U} \right\}.$$

Note that for all $k \in \{1, \ldots, m\}$, we have $mt_k \leq U$ and, thus, $s_m \geq s_{m-1}$.

We show that for any bag sizes $a_1, \ldots, a_m$, the adversary can force the algorithm to have a makespan of at least $\frac{U}{L}$ with some $\mathcal{S}_k$. Since the optimal

makespan is fixed to be equal to 1 by assumption, this implies a robustness factor of at least $\frac{U}{L}$.

Let $k^\star$ be the smallest index such that $a_k \geq \frac{t_k}{L}$. Such an index exists because the sum of the t_i's is equal to L (Eq. (1)) and the sum of the a_i's is equal to 1. Now, consider the speed configuration $\mathcal{S}_{k^\star}$. If one of the bags a_i for $i \geq k^\star$ is not scheduled on the m-th machine, then the makespan is at least $\frac{a_i}{s_1} \geq a_{k^\star}\frac{U}{t_{k^\star}} \geq \frac{U}{L}$. Otherwise, all a_i for $i \geq k^\star$ are scheduled on machine m. Then, using Eq. (1), the load on that machine is at least

$$\sum_{i \geq k^\star} a_i = 1 - \sum_{i<k^\star} a_i \geq 1 - \frac{1}{L}\sum_{i<k^\star} t_i = \frac{1}{L}\left(L - (m-1)t_{k^\star} + U - L\right) = \frac{U}{L}s_m\,.$$

Thus, either a machine $i < m$ with a bag $i' \geq k^*$ or machine $i = m$ has a load of at least $s_i \cdot \frac{U}{L}$ and determines the makespan. □

For given bag sizes, we call a speed configuration that maximizes the minimum makespan a *worst-case speed configuration*. Before we provide the strategy that obtains a matching robustness factor, we state a property of such best strategies for the adversary.

Lemma 2. *Given bag sizes and a worst-case speed configuration, for each machine i, there exists an optimal assignment of the bags to the machines such that only machine i determines the makespan.*

Note that, by Lemma 2, for a worst-case speed configuration, there are many different bag-to-machine assignments that obtain the same optimal makespan. Lemma 2 also implies that for such speed configurations all speeds are non-zero.

Let Sand denote the algorithm that creates m bags of the following sizes

$$a_1 = \frac{t_1}{L},\ a_2 = \frac{t_2}{L},\ \ldots\ ,\ a_m = \frac{t_m}{L}\,.$$

Note that this is a valid algorithm since the sum of these bag sizes is equal to 1. Moreover, these bag sizes are exactly such that if we take the speed configurations from Lemma 1, placing bag j on a slow machine in configuration j results in a makespan that is equal to $\bar{\rho}(m)$. We proceed to show that Sand ensures a robustness factor of $\bar{\rho}(m)$.

Lemma 3. *For any $m \geq 1$,* Sand *is $\bar{\rho}(m)$-robust for speed-robust scheduling with infinitely many infinitesimal jobs.*

Proof. Let $a_1, \ldots, a_m$ be the bag sizes as specified by Sand and let $s_1, \ldots, s_m$ be a worst-case speed configuration given these bag sizes. Consider an optimal assignment of bags to machines and let $C^*_{\max}$ denote its makespan. We use one particular (optimal) assignment to obtain an upper bound on $C^*_{\max}$. By Lemma 2, an optimal assignment exists where only machine 1 determines the makespan, i.e., machine 1 has load $C^*_{\max} \cdot s_1$ and any other machine i has load strictly less than $C^*_{\max} \cdot s_i$. Consider such an assignment. If there are two bags assigned to

machine 1, then there is an empty machine with speed at least s_1. Therefore, we can put one of the two bags on that machine and decrease the makespan. This contradicts that $C^*_{\max}$ is the optimal makespan, so there is exactly one bag assigned to machine 1. Let k be the index of the unique bag placed on machine 1, i.e., $C^*_{\max} = \frac{a_k}{s_1}$, and let ℓ be the number of machines of speed s_1.

If $a_k > a_\ell$, then machine $i \in \{1, \dots, \ell\}$, with speed s_1, can be assigned bag i with a load that is strictly less than $C^*_{\max} \cdot s_1$. Thus, given the current assignment, we can remove bag a_k from machine 1 and place the ℓ smallest bags on the ℓ slowest machines, one per machine, e.g., bag a_i on machine i for $i \in \{1, \dots, \ell\}$. This empties at least one machine of speed strictly larger than s_1. Then, we can place bag a_k on this (now empty) machine, which yields a makespan that is strictly smaller than $C^*_{\max}$. This contradicts the assumption that $C^*_{\max}$ is the optimal makespan, and thus, $a_k \leq a_\ell$, which implies $k \leq \ell$.

Let P_i denote the total processing time of bags assigned to machine i, and C the total *remaining capacity* of the assignment, that is, $C := \sum_{i=1}^{m} (s_i C^*_{\max} - P_i)$. We bound C, which allows us to bound $C^*_{\max}$.

Machines in the set $\{2, \dots, \ell\}$ cannot be assigned a bag of size larger than a_k since their load would be greater than $C^*_{\max} \cdot s_1$, causing a makespan greater than $C^*_{\max}$. Therefore, we assume without loss of generality that all bags $a_j < a_k$ are assigned to a machine with speed s_1. The total remaining capacity on the first k machines is therefore equal to $(k-1)a_k - \sum_{i<k} a_i$.

Consider a machine $i > k$. If the remaining capacity of this machine is greater than a_k, then we can decrease the makespan of the assignment by moving bag k to machine i. Therefore, the remaining capacity on machine i is at most a_k.

Combining the above and using (1), we obtain:

$$C \leq (m-1)a_k - \sum_{i<k} a_i = \frac{1}{L}\left((m-1)t_k - \sum_{i<k} t_i\right) = \frac{1}{L}\left(U - L\right) .$$

The total processing time is $\sum_{i=1}^{m} a_i = 1$, and the maximum total processing time that the machines could process with makespan $C^*_{\max}$ is equal to $\sum_{i=1}^{m} s_i C^*_{\max} = C^*_{\max}$. Since the latter is equal to the total processing time plus the remaining capacity, we have $C^*_{\max} = 1 + C \leq \frac{U}{L}$, which proves the lemma. □

The robustness factor $\bar{\rho}(m)$ is not best possible for every m when we allow algorithms that make randomized decisions and compare to an oblivious adversary. For $m = 2$, uniformly randomizing between bag sizes $a_1 = a_2 = \frac{1}{2}$ and $a_1 = \frac{1}{4}$, $a_2 = \frac{3}{4}$ yields a robustness factor that is slightly better than $\bar{\rho}(2) = \frac{4}{3}$. Interestingly, with speeds in $\{0, 1\}$ the optimal robustness factor is equal for deterministic and randomized algorithms.

2.2 Speeds in $\{0, 1\}$

Theorem 2. *For all $m \geq 1$, there is a deterministic $\bar{\rho}_{01}(m)$-robust algorithm for speed-robust scheduling with speeds in $\{0, 1\}$ and infinitesimal jobs, where*

$$\bar{\rho}_{01}(m) = \max_{t \in \mathbb{N},\ t \leq \frac{m}{2}} \frac{1}{\frac{t}{m-t} + \frac{m-2t}{m}} \leq \frac{1+\sqrt{2}}{2} = \bar{\rho}_{01} \approx 1.207 .$$

This is the best possible robustness factor that can be achieved by any algorithm, even by a randomized algorithm against an oblivious adversary.

The deterministic version of the lower bound and some useful insights were already presented in [20]. We recall some of these insights here because they are used in the proof. To do so, we introduce some necessary notation used in the remainder of this paper. The number of failing machines (i.e., machines with speed equal to 0) is referred to as t, with $t \in \{0, \dots, m-1\}$, and we assume w.l.o.g. that these are machines $1, \dots, t$. Furthermore, we assume for this subsection, again w.l.o.g., that the total volume of infinitesimal jobs is m, and we define bags $1, \dots, m$ with respective sizes $a_1 \leq \dots \leq a_m$ summing to at least m (the potential excess being unused).

Lemma 4 (Statement (3) in [20]). *For all $m \geq 1$ and $t \leq \frac{m}{2}$, there exists a makespan-minimizing allocation of bags to machines for speed-robust scheduling with speeds in $\{0, 1\}$ and infinitely many infinitesimal jobs that assigns the smallest $2t$ bags to machines $t+1, \dots, 2t$.*

Since Lemma 4 only works for $t \leq \frac{m}{2}$, one may worry that, for larger t, there is a more difficult structure to understand. The following insight shows that this worry is unjustified. Indeed, if $m' < \frac{m}{2}$ is the number of machines that do *not* fail, one can simply take the solution for $2m'$ machines, and assign the bags from any two machines to one machine. The optimal makespan is doubled and that of the algorithm is at most doubled, so the robustness is conserved.

Lemma 5 (Proof of Theorem 2.2 in [20]). *Let $\rho > 1$. For all $m \geq 1$, if an algorithm is ρ-robust for speed-robust scheduling with speeds in $\{0, 1\}$ and infinitely many infinitesimal jobs for $t \leq \frac{m}{2}$, it is ρ-robust for $t \leq m-1$.*

Therefore, we focus on computing bag sizes such that the makespan of a best allocation according to Lemma 4 is within a $\bar{\rho}_{01}(m)$ factor of the optimal makespan when $t \leq \frac{m}{2}$. The approach in [20] to obtain the (as we show tight) lower bound $\bar{\rho}_{01}(m)$ is as follows. Given some $t \leq \frac{m}{2}$ and a set of bags allocated according to Lemma 4,

(i) The makespan on machines $t+1, \dots, 2t$ is at most $\bar{\rho}_{01}(m)$ times the optimal makespan $\frac{m}{m-t}$, and
(ii) The makespan on machines $2t+1, \dots, m$ is a most $\bar{\rho}_{01}(m)$ because those machines only hold a *single* bag after a simple "folding" strategy for assigning bags to machines, which we define below.

In particular, since $t = 0$ is possible (ii) implies that all bag sizes are at most $\bar{\rho}_{01}(m)$. The fact that the total processing volume of m has to be accommodated and maximizing over t results in the lower bound given in Theorem 2.

To define bag sizes leading to a matching upper bound, we further restrict our choices when $t \leq \frac{m}{2}$ machines fail. Of course, as we match the lower bound, the restriction is no limitation but rather a simplification. When $t \leq \frac{m}{2}$ machines fail, we additionally assume that machines $t+1, \dots, 2t$ receive exactly two bags each:

Assuming $t \leq \frac{m}{2}$, the *simple folding* of these bags onto machines assigns bags $i \geq t+1$ to machine i, and bags $i \leq t$ (recall machine i fails) to machine $2t-i+1$. Hence, bags $1, \ldots, t$ are "folded" onto machines $2t, \ldots, t+1$ (sic).

For given m, let $t^\star$ be an optimal adversarial choice for t in Theorem 2. Assuming there are bag sizes $a_1, \ldots, a_m$ that match the bound $\bar{\rho}_{01}(m)$ through simple folding, by (i) and (ii), we precisely know the makespan on all machines after folding when $t = t^\star$. That fixes $a_i + a_{2t^\star+1-i} = \bar{\rho}_{01}(m) \cdot \frac{m}{m-t^\star}$ for all $i \in \{1, \ldots, t^\star\}$ and $a_{2t^\star+1} = \cdots = a_m = \bar{\rho}_{01}(m)$. In contrast to [20], we show that defining a_i for $i \in \{1, \ldots, t^\star\}$ to be essentially a linear function of i, and thereby fixing all bag sizes, suffices to match $\bar{\rho}_{01}(m)$. The word "essentially" can be dropped when replacing $\bar{\rho}_{01}(m)$ by $\bar{\rho}_{01}$.

A clean way of thinking about the bag sizes is through *profile functions* which reflect the distribution of load over bags in the limit case $m \to \infty$. Specifically, we identify the set $\{1, \ldots, m\}$ with the interval $[0, 1]$ and define a continuous non-decreasing profile function $\bar{f} : [0, 1] \to \mathbb{R}_+$ integrating to 1. A simple way of getting back from the profile function to actual bag sizes of total size approximately m is equidistantly "sampling" $\bar{f}$, i.e., by letting $a_i = \bar{f}\left(\frac{i-1/2}{m}\right)$ for all i.

Our profile function $\bar{f}$ implements the above observations and ideas in the continuous setting. Indeed, our choice

$$\bar{f}(x) = \min\left\{\frac{1}{2} + \bar{\rho}_{01} \cdot x, \bar{\rho}_{01}\right\} = \min\left\{\frac{1}{2} + \frac{(1+\sqrt{2}) \cdot x}{2}, \frac{1+\sqrt{2}}{2}\right\}$$

is linear up to $\beta = 2 - \sqrt{2} = \lim_{m\to\infty} \frac{2t^\star}{m}$ and from then on it is constantly $\bar{\rho}_{01} = \lim_{m\to\infty} \bar{\rho}_{01}(m)$. We give some intuition for why this function works using the continuous counterpart of folding: When $t \leq t^\star$ machines fail, i.e., a continuum of machines with measure $x \leq \frac{\beta}{2}$, we fold the corresponding part of $\bar{f}$ onto the interval $[x, 2x]$, yielding a rectangle of width x and height $\bar{f}(0) + \bar{f}(2x) = 2\bar{f}(x)$. We have to prove that the height does not exceed the optimal makespan $\frac{1}{1-x}$ by more than a factor of $\bar{\rho}_{01}$. Equivalently, we maximize $2\bar{f}(x)(1-x)$ (even over $x \in \mathbb{R}$) and observe the maximum of $\bar{\rho}_{01} = \frac{1+\sqrt{2}}{2}$ at $x = \frac{\beta}{2}$. When $x \in \left(\frac{\beta}{2}, \frac{1}{2}\right]$, note that by folding we *still* obtain a rectangle of height $2\bar{f}(x)$ (but width $\beta - x$), dominating the load on the other machines. Hence, the makespan is at most $\frac{\bar{\rho}_{01}}{1-x}$ for every $x \in \left[0, \frac{1}{2}\right]$.

Directly "sampling" $\bar{f}$, we obtain a weaker bound (stated below) than that in Theorem 2. The proof (and the algorithm) is substantially easier than that of the main theorem: Firstly, we translate the above continuous discussion into a discrete proof. Secondly, we exploit that $\bar{f}$ is concave to show that the total volume of the "sampled" bags is larger than m for every $m \in \mathbb{N}$. Later, we make use of the corresponding simpler algorithm. Let SAND$_{01}$ denote the algorithm that creates m bags of size $a_i := \bar{f}\left(\frac{i-1/2}{m}\right)$, for $i \in \{1, \ldots, m\}$.

Theorem 3. SAND$_{01}$ *is* $\bar{\rho}_{01}$*-robust for speed-robust scheduling with speeds in* $\{0, 1\}$ *and infinitely many infinitesimal jobs for all* $m \geq 1$.

As the profile function disregards specific machines, obtaining bag sizes through this function seems too crude to match $\bar{\rho}_{01}(m)$ for every m. Indeed, our proof of Theorem 2 is based on a much more careful choice of the bag sizes.

3 Speed-Robust Scheduling with Discrete Jobs

In this section, we consider the most general version of Speed Robust Scheduling. While in Sects. 2 and 4 we crucially use in our algorithm design the assumptions that all jobs are infinitesimally small (sand) or are of the same size (bricks), respectively, here, their sizes can vary arbitrarily (rocks).

By a scaling argument, we may assume w.l.o.g. that the machine speeds satisfy $\sum_{i=1}^{m} s_i = \sum_{j=1}^{n} p_j$. Observe that minimizing the size of a largest bag may not yield a robust algorithm.

Lemma 6. *Algorithms for speed-robust scheduling that minimize the size of a largest bag may not have a constant robustness factor.*

Proof. Consider any integer $k \geq 1$, a number of machines $m = k^2 + 1$, one job with processing time k, and k^2 unit-size jobs. The maximum bag size is at least k, so an algorithm that builds $k + 1$ bags of size k respects the conditions of the lemma. Consider the speed configuration where k^2 machines have speed 1 and one machine has speed k. It is possible to schedule all jobs on these machines with makespan 1. However, the algorithm must either place a bag on a machine of speed 1 or all bags on the machine of speed k, which gives a makespan of k. □

For machine speeds in $\{0, 1\}$, such algorithms are $\left(2 - \frac{2}{m}\right)$-robust. Once the number m' of speed-1 machines is revealed, simply combine the two smallest bags repeatedly if $m' < m$. The makespan is then at most twice the average load on $m' + 1$ machines, i.e., $\frac{2m'}{m'+1}$ times the average load on m' machines.

The lower bound in Lemma 6 exploits the fact that bags sizes of such algorithms might be very unbalanced. An algorithm is called *balanced* if, for an instance of unit-size jobs, the bag sizes created by the algorithm differ by at most one unit. In particular, a balanced algorithm creates m bags of size k when confronted with mk unit-size jobs and m bags. For balanced algorithms, we give a lower bound in Lemma 7 and a matching upper bound in Theorem 4.

Lemma 7. *No balanced algorithm for speed-robust scheduling can obtain a better robustness factor than* $2 - \frac{1}{m}$ *for any* $m \geq 1$.

We now show that this lower bound is attained by a simple algorithm, commonly named as *Longest Processing Time first* (LPT) which considers jobs in non-increasing order of processing times and assigns each job to the bag that currently has the smallest size, i.e., the minimum allocated processing time.

Theorem 4. LPT *is* $\left(2 - \frac{1}{m}\right)$*-robust for speed-robust scheduling for all* $m \geq 1$.

Proof. While we may assume that the bags are allocated optimally to the machines once the speeds are given, we use a different allocation for the analysis. This cannot improve the robustness.

Consider the m bags and let b denote the size of a largest bag, B, that consists of at least two jobs. Consider all bags of size strictly larger than b, each containing only a single job, and place them on the same machine as Opt places the corresponding jobs. We define for each machine i with given speed s_i a capacity bound of $\left(2-\frac{1}{m}\right)\cdot s_i$. Then, we consider the remaining bags in non-increasing order of bag sizes and iteratively assign them to the – at the time of assignment – least loaded machine with sufficient remaining capacity.

By the assumption $\sum_{i=1}^{m} s_i = \sum_{j=1}^{n} p_j$and the capacity constraint $\left(2-\frac{1}{m}\right)\cdot s_i$, it is sufficient to show that LPT can successfully place all bags.

The bags larger than b fit by definition as they contain a single job. Assume by contradiction that there is a bag which cannot be assigned. Consider the first such bag and let T be its size. Let $k < m$ be the number of bags that have been assigned already. Further, denote by w the size of a smallest bag. Since we used LPT to create the bags, we have $w \geq \frac{1}{2}b$. To see that, consider bag B and notice that the smallest job in it has a size at most $\frac{1}{2}b$. When this job was assigned to its bag, B was the bag with smallest size, and this size was at least $\frac{1}{2}b$ since we allocate jobs in LPT-order. Hence, the size of a smallest bag is $w \geq \frac{1}{2}b \geq \frac{1}{2}T$, where the second inequality is true as all bags larger than b can be placed.

We use this inequality to give a lower bound on the total remaining capacity on the m machines when the second-stage algorithm fails to place the $(k+1)$-st bag. The $(m-k)$ bags that were not placed have a combined volume of at least $V_\ell = (m-k-1)w + T \geq (m-k+1)\frac{T}{2}$. The bags that were placed have a combined volume of at least $V_p = kT$. The remaining capacity is then at least $C = (2-\frac{1}{m})V_\ell + (1-\frac{1}{m})V_p$, and we have

$$\begin{aligned} C &= \left(2-\frac{1}{m}\right)V_\ell + \left(1-\frac{1}{m}\right)V_p \geq \left(2-\frac{1}{m}\right)(m-k+1)\frac{T}{2} + \left(1-\frac{1}{m}\right)kT \\ &\geq (m-k+1)T - (m-k+1)\frac{T}{2m} + kT - \frac{1}{m}kT \geq mT + T - \frac{m+k+1}{2m}T \\ &\geq mT\,. \end{aligned}$$

Thus, there is a machine with remaining capacity T which contradicts the assumption that the bag of size T does not fit. □

4 Speed-Robust Scheduling with Equal-Size Jobs

In this section we consider instances where all jobs are of equal size, i.e., bricks, as this case seems to capture the complexity of the general problem. This intuition stems from the fact that all known lower bounds already hold for this type of instances (see [20] and Lemma 10).

By a scaling argument, we may assume that all jobs have unit processing time. Before focusing on a specific speed setting, we show that in both settings we can

use any algorithm for infinitesimal jobs with proper scaling to obtain a robustness which is degraded by a factor decreasing with $\frac{n}{m} =: \lambda$. Assume $\lambda > 1$, as otherwise the problem is trivial. We define the algorithm SandForBricks that builds on the optimal algorithm for infinitesimal jobs, Sand*, which is Sand for general speeds (Sect. 2.1) or Sand$_{01}$ for speeds in $\{0, 1\}$ (Sect. 2.2). Let $a_1, \ldots, a_m$ be the bag sizes constructed by Sand* scaled such that a total processing volume of n can be assigned, that is, $\sum_{i=1}^{m} a_i = n$. For unit-size jobs, we define bag sizes as $a'_i = \left(1 + \frac{1}{\lambda}\right) \cdot a_i$ and assign the jobs greedily to the bags.

Lemma 8. *For n jobs with unit processing times and m machines,* SandForBricks *for speed-robust scheduling is $\left(1 + \frac{1}{\lambda}\right) \cdot \rho(m)$-robust, where $\lambda = \frac{n}{m}$ and $\rho(m)$ is the robustness factor for* Sand* *for m machines.*

4.1 General Speeds

For bricks, i.e., unit-size jobs, we beat the factor $2 - \frac{1}{m}$ (Theorem 4) for speed-robust scheduling and give a 1.8-robust algorithm. For $m = 2$ and $m = 3$, we give algorithms with best possible robustness factors $\frac{4}{3}$ and $\frac{3}{2}$, respectively.

Theorem 4 shows that LPT has a robustness factor of $2 - \frac{1}{m}$. We show that a slightly different algorithm, BuildOdd, has a robustness that increases with the ratio between the number of jobs and the number of machines. BuildOdd builds bags of three possible sizes: for $q \in \mathbb{N}$ such that $\lambda = \frac{n}{m} \in [2q-1, 2q+1]$, the bags sizes are $2q-1$, $2q$ and $2q+1$. In a manner similar to the proof of Theorem 4, we can prove BuildOdd is $(2 - \frac{1}{q+1})$-robust. The worst case happens when a bag of size $2q+1$ is scheduled on a machine of speed $q+1$.

Lemma 9. *For n unit-size jobs, m machines and $q \in \mathbb{N}$ with $\lambda \in [2q-1, 2q+1]$,* BuildOdd *is $(2 - \frac{1}{q+1})$-robust for speed-robust scheduling.*

The robustness guarantees in Lemmas 8 and 9 are decreasing and increasing, respectively, in λ. By carefully choosing between BuildOdd and SandForBricks, depending on the input, we obtain an improved algorithm for bricks. For $\lambda < 8$, we execute BuildOdd, which yields a robustness factor of at most 1.8 by Lemma 9, as $q \leq 4$ for $\lambda < 8$. Otherwise, when $\lambda \geq 8$, we run SandForBricks with a guarantee of $\frac{9}{8} \cdot \frac{e}{e-1} \approx 1.78$ by Lemma 8.

Theorem 5. *There is an algorithm for speed-robust scheduling with unit-size jobs that has a robustness factor of at most 1.8 for any $m \geq 1$.*

We give a general lower bound on the best achievable robustness factor.

Lemma 10. *For every $m \geq 3$, no algorithm for speed-robust scheduling can have a robustness factor smaller than $\frac{3}{2}$, even restricted to unit-size jobs.*

For special cases with few machines, we give best possible algorithms.

Lemma 11. *An optimal algorithm for speed-robust scheduling for unit-size jobs has robustness factor $\frac{4}{3}$ on $m = 2$ machines and $\frac{3}{2}$ on $m = 3$ machines, and larger than $\bar{\rho}(6) > \frac{3}{2}$ for $m = 6$.*

4.2 Speeds in $\{0, 1\}$

When considering speeds in $\{0, 1\}$, bricks (unit-size jobs) are of particular interest as the currently best known lower bound for rocks (arbitrary jobs) is $\frac{4}{3}$ and uses only bricks [20]. We present an algorithm with a matching upper bound.

Theorem 6. *There exists a $\frac{4}{3}$-robust algorithm for speed-robust scheduling with $\{0, 1\}$-speeds and unit-size jobs.*

In the proof, we handle different cases depending on m and $\lceil\lambda\rceil$ by carefully tailored methods. Note that $\lceil\lambda\rceil$ is equal to the optimal makespan on m machines.

When $\lceil\lambda\rceil \geq 11$, we use SandForBricks and obtain a robustness factor of at most $\frac{4}{3}$ by Lemma 9. The proof uses a volume argument to show that jobs fit into the scaled optimal bag sizes for infinitesimal jobs, even after rounding bag sizes down to the nearest integer. When $\lceil\lambda\rceil \in \{9, 10\}$, this method is too crude. We refine it to show that for $m \geq 40$ it is still possible to scale bag sizes from SandID and round them to integral sizes such that all jobs can be placed. The analysis exploits an amortized bound on the loss due to rounding over consecutive bags. For the case that $\lceil\lambda\rceil \leq 8$ and $m \geq 50$, we use a constructive approach and give a strategy that utilizes at most four different bag sizes. The remaining cases, $\lceil\lambda\rceil \leq 10$ and $m \leq 50$, can be verified by enumerating over all possible instances and using an integer linear program to verify that there is a solution of bag sizes that is $\frac{4}{3}$-robust.

References

1. 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
2. Albers, S., Schmidt, G.: Scheduling with unexpected machine breakdowns. Discret. Appl. Math. **110**(2–3), 85–99 (2001). https://doi.org/10.1016/s0166-218x(00)00266-3
3. Alon, N., Azar, Y., Woeginger, G.J., Yadid, T.: Approximation schemes for scheduling on parallel machines. J. Sched. **1**(1), 55–66 (1998). https://doi.org/10.1002/(sici)1099-1425(199806)1:1⟨55::aid-jos2⟩3.0.co;2-j
4. Baruah, S.K., et al.: Scheduling real-time mixed-criticality jobs. IEEE Trans. Comput. **61**(8), 1140–1152 (2012). https://doi.org/10.1109/tc.2011.142
5. Chen, L., Megow, N., Rischke, R., Stougie, L.: Stochastic and robust scheduling in the cloud. In: APPROX-RANDOM. LIPIcs, vol. 40, pp. 175–186. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2015). https://doi.org/10.4230/LIPIcs.APPROX-RANDOM.2015.175
6. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008). https://doi.org/10.1145/1327452.1327492
7. Diedrich, F., Jansen, K., Schwarz, U.M., Trystram, D.: A survey on approximation algorithms for scheduling with machine unavailability. In: Lerner, J., Wagner, D., Zweig, K.A. (eds.) Algorithmics of Large and Complex Networks. LNCS, vol. 5515, pp. 50–64. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02094-0_3

8. 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
9. Eberle, F., Hoeksma, R., Megow, N., Nölke, L., Schewior, K., Simon, B.: Speed-robust scheduling. CoRR (2020). https://arxiv.org/abs/2011.05181
10. Epstein, L., et al.: Universal sequencing on an unreliable machine. SIAM J. Comput. **41**(3), 565–586 (2012). https://doi.org/10.1137/110844210
11. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell Syst. Tech. J. **45**(9), 1563–1581 (1966). https://doi.org/10.1002/j.1538-7305.1966.tb01709.x
12. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems theoretical and practical results. J. ACM **34**(1), 144–162 (1987). https://doi.org/10.1145/7531.7535
13. Jansen, K.: An EPTAS for scheduling jobs on uniform processors: using an MILP relaxation with a constant number of integral variables. SIAM J. Discrete Math. **24**(2), 457–485 (2010). https://doi.org/10.1137/090749451
14. Kouvelis, P., Yu, G.: Robust Discrete Optimization and Its Applications. Springer, Berlin (1997). https://doi.org/10.1007/978-1-4757-2620-6
15. Levi, R., Magnanti, T.L., Shaposhnik, Y.: Scheduling with testing. Manag. Sci. **65**(2), 776–793 (2019). https://doi.org/10.1287/mnsc.2017.2973
16. Megow, N., Verschae, J.: Dual techniques for scheduling on a machine with varying speed. SIAM J. Discret. Math. **32**(3), 1541–1571 (2018). https://doi.org/10.1137/16m105589x
17. Niño-Mora, J.: Stochastic scheduling. In: Encyclopedia of Optimization, pp. 3818–3824. Springer (2009). https://doi.org/10.1007/978-0-387-74759-0_665
18. Pruhs, K., Sgall, J., Torng, E.: Online scheduling. In: Handbook of Scheduling. Chapman and Hall/CRC (2004). https://doi.org/10.1007/978-3-319-99849-7
19. Shmoys, D.B., Sozio, M.: Approximation algorithms for 2-stage stochastic scheduling problems. In: Fischetti, M., Williamson, D.P. (eds.) IPCO 2007. LNCS, vol. 4513, pp. 145–157. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72792-7_12
20. Stein, C., Zhong, M.: Scheduling when you do not know the number of machines. ACM Trans. Algorithms **16**(1), 9:1–9:20 (2020). https://doi.org/10.1145/3340320