Speed-Robust Scheduling Revisited

Josef Minařík
Computer Science Institute of Charles Univ., Faculty of Mathematics and Physics, Prague, Czechia

Jiří Sgall
Computer Science Institute of Charles Univ., Faculty of Mathematics and Physics, Prague, Czechia

Abstract

Speed-robust scheduling is the following two-stage problem of scheduling n jobs on m uniformly related machines. In the first stage, the algorithm receives the value of m and the processing times of n jobs; it has to partition the jobs into b groups called bags. In the second stage, the machine speeds are revealed and the bags are assigned to the machines, i.e., the algorithm produces a schedule where all the jobs in the same bag are assigned to the same machine. The objective is to minimize the makespan (the length of the schedule). The algorithm is compared to the optimal schedule and it is called ρ-robust, if its makespan is always at most ρ times the optimal one.

Our main result is an improved bound for equal-size jobs for $b = m$. We give an upper bound of 1.6. This improves previous bound of 1.8 and it is almost tight in the light of previous lower bound of 1.58. Second, for infinitesimally small jobs, we give tight upper and lower bounds for the case when $b \geq m$. This generalizes and simplifies the previous bounds for $b = m$. Finally, we introduce a new special case with relatively small jobs for which we give an algorithm whose robustness is close to that of infinitesimal jobs and thus gives better than 2-robust for a large class of inputs.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Scheduling algorithms

Keywords and phrases scheduling, approximation algorithms, makespan, uniform speeds

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2024.8

Category APPROX

Related Version *Full Version*: https://arxiv.org/abs/2407.11670 [8]

Funding Partially supported by grant 24-10306S of GA ČR.

Acknowledgements We are grateful to Franziska Eberle for insightful discussions and comments, and to the anonymous referees for detailed and useful reviews.

1 Introduction

Speed-robust scheduling is a two-stage problem that was introduced by Eberle *et al.* [4]. The eventual goal is to schedule on m uniformly related machines, however their speeds are not known at the beginning. In the first stage, the algorithm receives the value of m and the processing times of n jobs; it has to partition the jobs into b groups called bags. In the second stage, the machine speeds are revealed and the bags are assigned to the machines, i.e., the algorithm produces a schedule where all the jobs in the same bag are assigned to the same machine. The objective is to minimize the makespan (the length of the schedule). The algorithm is compared to the optimal schedule of the jobs on the machines with known speeds; it is called ρ-robust, if its makespan is always at most ρ times the optimal one.

This problem is motivated by situations like the following one. Suppose that you have n computational tasks that you want to solve. You have a computational cluster available, but with unknown parameters. You only know that there will be (at most) m machines available on the cluster. You do not know anything about the performance of the machines – some of the machines might be faster than others; you only know that there will be (at most) m

machines available on the cluster. Furthermore, you can submit at most b different tasks to the cluster. Hence you will have to partition your n tasks into at most b groups. One such group will then have to be executed on one machine. The cluster will then schedule the groups optimally, knowing the speeds of the machines, and minimize the makespan.

Studying uncertainty in scheduling has a long history. In the classical online scheduling [10], the machine environment is usually fixed and the uncertainty stems from job arrivals. Considering uncertainty in the machine environment is less frequent. One early example is the work of Csanád and Noga [7], where additional machines can be bought for a certain cost. A substantial body of research with changing machine speeds is the area of dynamic speed scaling, in particular in the context of minimizing the power consumption, see [1, 9]; however, note that here the changing speeds are not a part of the adversarial environment but used by the algorithm to its advantage. Another direction considers online scheduling with unavailability periods [3]. One-machine scheduling with adversarially changing machine speed was considered in [5] in the context of unreliable machines.

Completely reversing the scenario with all jobs known from the beginning but uncertain machine environment is a recent new model introduced by Stein and Zhong [11] and Eberle et al. [4], see Section 1.2.

1.1 Formal definitions

Formally, in the first stage, we receive three positive integers n, m, b and n non-negative real numbers $p_1, \ldots, p_n$ representing the processing times of n jobs. The total processing time is denoted $P = \sum_{j=1}^{m} p_j$. The output of our first-stage algorithm is a mapping $B : \{1, \ldots, n\} \to \{1, \ldots, b\}$, where $B(j) = i$ represents the fact that the job j was assigned to the bag i. The sum of the processing times of all the jobs assigned to bag i the *size* of bag i and denoted $a_i = \sum_{j:B(j)=i} p_j$. The exact mapping B is not important for the second stage since the makespan depends only on the bag sizes.

In the second stage, we are given the bag sizes $a_1, \ldots, a_b$ and the previously unknown machine speeds $s_1, \ldots, s_m \geq 0$, not all equal to 0. We partition the bag indices $\{1, \ldots, b\}$ into m sets $M_1, \ldots, M_m$, representing the assignment to the m machines. Machine i then has a completion time $C_i = (\sum_{j \in M_i} a_j)/s_i$; for $s_i = 0$ we require $M_i = \emptyset$ and set $C_i = 0$, i.e., machine of speed 0 does not accept any jobs. Finally, $C_{\max} = \max_{i=1}^{m} C_i$ is the makespan, i.e., the length of the schedule.

Let $C^*_{\max}$ denote the makespan of the adversary, who does not have to create bags and can assign jobs directly to machines. Alternatively and equivalently, the adversary also creates bags, but with the knowledge of the speeds already in the first stage.

We call a first stage algorithm *ρ-robust* if, for all possible inputs and for all possible choices of machine speeds, there exists a second-stage assignment of bags to machines such that $C_{\max} \leq \rho \cdot C^*_{\max}$. Intuitively, an algorithm is ρ-robust if it performs at most ρ times worse than the adversary.

The previous definition implicitly assumes that the second stage is solved optimally. This is reasonable, as the scheduling on uniformly related machines allows PTAS, see [6, 12], so the chosen (presumably optimal) second-stage solution can be replaced by an arbitrarily good approximation. Also, our proofs show that the second-stage algorithm can be implemented by efficient greedy algorithms without any loss of performance, once the optimal makespan or its approximation is known.

We call the special cases of the problem *sand*, *bricks*, *rocks* and *pebbles*. Sand, bricks, and rocks were introduced by Eberle et al. [4]. These words represent the types of jobs.

- Rocks can be any shape or size and represent jobs of arbitrary processing time. This is the most general setting.
- Bricks are all the same and represent jobs with equal processing times.
- Sand grains are very small and represent infinitesimally small processing times.
- Pebbles represent jobs that are relatively small compared to the average load of all machines. We call an instance of speed-robust scheduling q-pebbles if $p_j \le q \cdot \frac{P}{m}$ holds for all jobs j.

1.2 Previous results

The two-stage scheduling problem with uncertainty in the machine environment was introduced by Stein and Zhong [11]. They focused on the case of m identical machines where in the second stage some machine might fail and then do not process any tasks. This amounts to a special case of speed-robust scheduling where $s_i \in \{0, 1\}$ for $1 \le i \le m$. They gave lower bounds of 4/3 for equal-size jobs (bricks) and $(\sqrt{2}+1)/2 \approx 1.207$ for infinitesimal jobs (sand). Their algorithms were later improved by Eberle *et al.* [4] to algorithms matching the lower bounds in both cases.

Our immediate predecessor, Eberle *et al.* [4], introduced the speed-robust scheduling for general speeds, i.e., on uniformly related machines. They studied mainly the case $b = m$, i.e., the case when the number of bags is equal to the number of machines. For this case they gave tight bounds for sand for every m, for large m the bound approaches $e/(e-1) \approx 1.58$. For equal-size jobs (bricks), they have shown an upper bound of 1.8.

For the most general case of rocks, the strongest known result is the algorithm with the robustness factor at most $1 + (m-1)/b$, which equals $2 - 1/m$ for $b = m$, given also by Eberle *et al.* [4]. It remains an interesting open problem to improve this bound, in particular to give an upper bound $2 - \varepsilon$ for rocks and $b = m$.

1.3 Our results

We now describe our results and compare them to the previous ones in each of the scenarios.

Sand. For sand, we give matching lower and upper bounds for any b and m. Namely, for $b \ge m$ we give an optimal algorithm which is $\overline{\rho}(m, b)$-robust for

$$\overline{\rho}(m, b) = \frac{m^b}{m^b - (m-1)^b} = \frac{1}{1 - \left(1 - \frac{1}{m}\right)^b} . \tag{1.1}$$

This matches the results of Eberle *et al.* [4] who gave an algorithm with the robustness factor equal to $\overline{\rho}(m, b) \le e/(e-1) \approx 1.58$ for $b = m$, generalizes them to arbitrary $b \ge m$ and significantly simplifies the proof.

An interesting case is when the number of bags is a constant multiple of m. For a fixed $\alpha \ge 1$ and $b = \alpha m$, our bound approaches $1/(1 - e^{-\alpha})$ from below for a large m. For example, doubling the number of bags to $b = 2m$ decreases the robustness factor from 1.58 to 1.16.

If $b < m$, the second-stage algorithm uses only the b fastest machines, so we can decrease m to $m' = b$ and tight results with robustness factor $\overline{\rho}(m', b) = \overline{\rho}(b, b)$ follow already from [4].

Pebbles. For the new case of q-pebbles and $b \ge m$, we give a $(\overline{\rho}(m, b) + q)$-robust algorithm. For $p < 0.42$, this gives an algorithm with the robustness factor below 2, i.e., below the currently strongest known upper bound for rocks.

Bricks. As our main result, we give a 1.6-robust algorithm for bricks for $b = m$. This improves the bound of 1.8 from Eberle *et al.* [4].

Furthermore, as a direct application of our results for pebbles we give a $(\overline{\rho}(m, b) + m/n)$-robust algorithm for any n and $b \geq m$. This improves and generalizes the $((1+m/n)\overline{\rho}(m, m))$-robust algorithm for $b = m$ given by Eberle *et al.* [4]. Namely, we improve the multiplicative factor of $(1 + m/n)$ to only an additive term of m/n.

Structure of the paper. We give some general preliminaries in Section 2. We give the results for sand and pebbles in Sections 3 and 4. We focus on our main result for bricks in Section 5. Some small cases need computer verification or tabulation of parameters, results of these are given in the full version of the paper on arXiv [8].

2 Preliminaries

We assume that the processing times, the machine speeds, and the bag sizes are always listed in a non-increasing order.

In the rest of this paper, we will make two assumptions below that restrict the speeds to particular special cases. This is without loss of generality, leveraging the fact that the algorithm must commit the bag sizes in the first phase without knowing the speeds.

- The optimal makespan is equal to 1. This implies that the robustness factor is equal to the makespan of the algorithm.
  Scaling all the speeds does not change the ratio of the makespans of our algorithm and the adversary. Thus for every instance of the problem, there exists another instance with $C^*_{\max} = 1$ that differs only in the speeds and the ratio of makespans of our algorithm and the adversary remains the same. It follows that any first-stage algorithm that is ρ-robust for instances with $C^*_{\max} = 1$ is ρ-robust for general instances, too.
- The sum of the processing times of all jobs equals to the sum of the speeds of all the machines, i.e., $P = \sum_{i=1}^{m} p_i = \sum_{i=1}^{m} s_i$. In other words, the adversary is fully utilizing all the machines, and the completion time of all the machines with non-zero speed is equal to 1, using the previous assumption.
  If there is some machine i with $s_i > 0$ and completion time $C^*_i < 1$ in the optimal schedule, we change its speed to $s'_i = C^*_i s_i$. This does not change the optimal makespan of the adversary and the makespan of the algorithm can only increase. Once again, it follows that any first-stage algorithm that is ρ-robust for these special instances is also ρ-robust for general instances.

For the second stage, typically, we use a simple greedy algorithm for the second stage instead of analyzing the optimal schedule. Technically, for an algorithm we need to know the optimal makespan (to modify the speeds appropriately, according to the assumptions above). However note that first we can approximate the makespan and second the algorithm is only used as a tool in the analysis.

For sand and pebbles we use Algorithm GreedyAssignment (see below), a variant of the well-known LPT algorithm. It is parameterized by ρ, the robustness factor to be achieved. At the beginning, every machine is assigned a capacity equal to its speed multiplied by ρ. The algorithm then goes through all the bags from large to small, assigns them on the machine with the largest capacity remaining, and decreases the capacity appropriately. If the capacities remain non-negative at the end, the makespan of the created assignment is at most ρ since machine i has been assigned jobs of total processing times at most ρs_i.

We use this to formulate the following sufficient condition for ρ-robustness of an algorithm which is instrumental in proving the upper bounds for sand and pebbles.

Algorithm GreedyAssignment.

Input: bag sizes $a_1 \geq \cdots \geq a_b$; machine speeds $s_1 \geq \cdots \geq s_m$; desired robustness factor ρ
for $i \leftarrow 1$ to m **do**
  $c_i \leftarrow \rho s_i$ ▷ Initialize the capacities of all machines
  $M_i = \emptyset$ ▷ Initialize the assignment
for $k \leftarrow 1$ to b **do**
  $i \leftarrow$ index of a machine with the largest c_i
  $M_i \leftarrow M_i \cup \{k\}$ ▷ Assign bag k to machine i
  $c_i \leftarrow c_i - a_k$ ▷ Decrease the remaining capacity of the selected machine
return $M_1, \ldots, M_m$

▶ **Theorem 2.1.** *If a first-stage algorithm always produces bag sizes satisfying inequalities* $a_k \leq \dfrac{\rho P - \sum_{j=1}^{k-1} a_j}{m}$*, for all* $k = 1, \ldots, b$*, then the algorithm is* ρ*-robust.*

Proof. Recall that we assume $\sum_{i=1}^{m} s_i = P$. We claim that the second-stage algorithm GreedyAssignment produces an assignment with makespan at most ρ. We only need to show that there is a machine with capacity at least a_k when assigning the kth bag. The initial total capacity was ρP and was already decreased by $\sum_{j=1}^{k-1} a_j$ at the time of assigning bag a_k. It follows that the remaining capacity is equal to $\rho P - \sum_{j=1}^{k-1} a_j$ and thus there exists a machine with capacity at least $(\rho P - \sum_{j=1}^{k-1} a_j)/m \geq a_k$. ◀

3 Sand

Intuitively, the case of sand corresponds to the limit case where n is large and all the jobs are small and have equal sizes. One can view this as an infinite number of infinitesimal jobs.

More formally, we are given just m, b, and P as an input of the first stage. The result of the first stage are b non-negative reals $a_1, \ldots, a_b$ whose sum equals P. The formulation of the second stage remains the same.

The model of infinitesimally small jobs resembles preemptive scheduling. In the optimal algorithm for preemptive scheduling [2], one needs to maintain the loads of machines in a geometric sequence with common ratio $m/(m-1)$ for m machines, roughly speaking. The proofs for sand show that here the same geometric sequence is also crucial, in particular it is used for the bag sizes in the algorithm. We now describe the sequence and state its properties useful both for the upper and lower bounds.

We define $U = m^b$, $L = m^b - (m-1)^b$ and $t_j = m^{b-j}(m-1)^{j-1}$ for $j \in \{1, \ldots, b\}$.

Observe that equation (1.1) defines $\overline{\rho}$ as $\overline{\rho}(m, b) = U/L$.

▶ **Lemma 3.1.** *For all* $k = 1, \ldots, b$*, it holds that* $\sum_{j=1}^{k} t_j = U - (m-1)t_k$*. In particular,* $\sum_{j=1}^{b} t_j = U - (m-1)^b = L$*.*

Proof. We proceed by induction on k. The lemma holds for $k = 1$ since $U = mt_1$ and thus $t_1 = U - (m-1)t_1$. Now suppose it holds for k. We can derive

$$\sum_{j=1}^{k+1} t_j = U - (m-1)t_k + t_{k+1} = U - m^{b-k}(m-1)^k + m^{b-k-1}(m-1)^k = U - (m-1)t_{k+1}$$

which completes the induction step. ◀

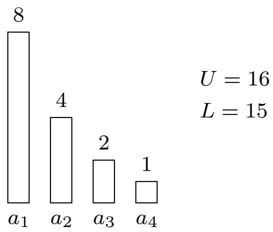

Figure 3.1 An example of bag sizes chosen for $m = 2$ and $b = 4$.

To get some intuition behind the algorithm for sand, it might be useful to consider the case $m = 2$, see Figure 3.1. Suppose that $P = L = 2^b - 1$, choose the bag sizes $a_k = t_k = 2^{b-k}$. For $m = 2$ the sizes are powers of two, so it is easy to see that we can achieve the robustness ratio of $1 + 1/(2^b - 1) = 2^b/(2^b - 1)$ as follows: The adversary chooses any speeds s_1, s_2 such that $s_1 + s_2 = P = 2^b - 1$. The capacities of the machines (as in GreedyAssignment) then satisfy $c_1 + c_2 = 2^b$ and thus $\lfloor c_1 \rfloor + \lfloor c_2 \rfloor \geq 2^b - 1$. We can express $\lfloor c_1 \rfloor$ in binary, assign the corresponding bags on the first machine and the remaining bags to the second machine.

3.1 Upper bound

We use a different approach than Eberle et al. [4] for the proof of the upper bound. We choose the same bag sizes (for $b = m$) but we simplify the proof by use of Theorem 2.1. Algorithm Sand describes the bag sizes. Note that the sum of bag sizes produced by Sand is P, using Lemma 3.1.

Algorithm Sand.

Input: number of bags b; number of machines m; total amount of sand P
$L \leftarrow m^b - (m-1)^b$
for $j \leftarrow 1$ to b **do** $a_j \leftarrow t_j \frac{P}{L}$
return $a_1, a_2, \ldots, a_b$

▶ **Theorem 3.2.** *Algorithm* Sand *is* $\overline{\rho}(m, b)$*-robust for sand, for* $\overline{\rho}$ *defined by* (1.1)*.*

Proof. We assume $P = L$ since it does not change the ratio of our makespan and the makespan of the adversary. Under this assumption, Sand produces bag sizes $a_k = t_k$.

It is sufficient to show that the bag sizes produced by Sand satisfy the condition of Theorem 2.1. Let us prove the kth inequality in the assumption of the theorem. We have

$$\overline{\rho}(m, b)P - \sum_{j=1}^{k-1} a_j = \frac{U}{L} L - \sum_{j=1}^{k-1} t_j = U - \sum_{j=1}^{k} t_j + t_k \,.$$

According to Lemma 3.1, we can simplify the right-hand side as follows.

$$U - \sum_{j=1}^{k} t_j + t_k = U - (U - (m-1)t_k) + t_k = mt_k = ma_k \,.$$

The kth inequality in the assumption of Theorem 2.1 follows, in fact it holds with equality.

Theorem 2.1 now implies that there exists an assignment with makespan at most $\overline{\rho}(m, b)$. ◀

3.2 Lower bound

The following proof is a slightly modified and generalized version of the proof by Eberle et al. [4]. The main difference is that we do not require the number of bags and machines to be the same.

▶ **Theorem 3.3.** *No deterministic algorithm for sand may have a robustness factor smaller than $\overline{\rho}(m, b)$, for $\overline{\rho}$ defined by* (1.1)*.*

Proof. Let us without loss of generality assume $P = U$ (be aware that we assumed $P = L$ in the proof of the upper bound). Let us denote the chosen bag sizes by $a_1 \geq \cdots \geq a_b$. We will restrict the adversary to b different speed configurations indexed by k, where

$$\mathcal{S}_k = \{s_1 = U - (m-1)t_k,\ s_2 = t_k,\ s_3 = t_k,\ \ldots,\ s_m = t_k\}\,.$$

See Figure 3.2 for an example. Note that the sum of machine speeds is equal to U in every configuration and hence the makespan of the adversary is indeed 1 as we always assume. In every speed configuration, there are $m-1$ slow machines and one fast machine, since

$$s_1 = U - (m-1)t_k = \sum_{j=1}^{k} t_j \geq t_k\,.$$

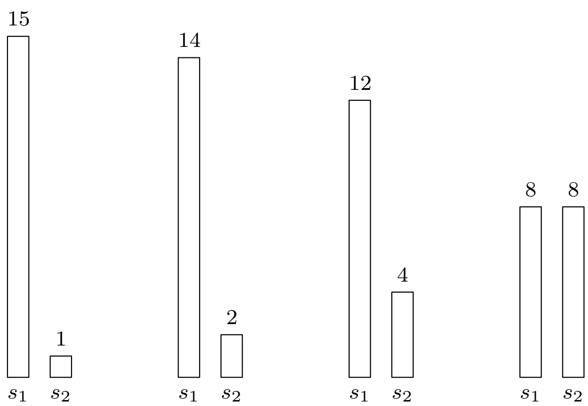

Figure 3.2 An example of speed configurations considered by the adversary for $m = 2$ and $b = 4$.

Let $k_{\max}$ be the largest index such that $a_{k_{\max}} \geq \frac{U}{L} t_{k_{\max}}$. This index must exist since

$$\sum_{j=1}^{b} a_j = U = \frac{U}{L} L = \frac{U}{L} \sum_{j=1}^{b} t_j\,.$$

Now let the adversary choose the speed configuration $\mathcal{S}_{k_{\max}}$. We distinguish two cases depending on the bag assignment in the second stage.

Case 1. At least one of the bags $a_1, \ldots, a_{k_{\max}}$ is assigned to a slow machine. The makespan is at least the completion time of this machine which is at least

$$\frac{a_j}{t_{k_{\max}}} \geq \frac{a_{k_{\max}}}{t_{k_{\max}}} \geq \frac{U}{L}\,.$$

Case 2. All of the bags $a_1, \ldots, a_{k_{\max}}$ are assigned to the fast machine. Total size of the bags assigned to the fast machine is at least

$$\sum_{j=1}^{k_{\max}} a_j = U - \sum_{j=k_{\max}+1}^{b} a_j \,.$$

By definition of $k_{\max}$ it holds that $a_j < \frac{U}{L} t_j$ for $j > k_{\max}$ and we can bound

$$U - \sum_{j=k_{\max}+1}^{b} a_j \geq U - \frac{U}{L} \sum_{j=k_{\max}+1}^{b} t_j \,.$$

Since $\sum_{j=1}^{b} t_j = L$, we can rearrange the right-hand side as follows

$$U - \frac{U}{L} \sum_{j=k_{\max}+1}^{b} t_j = U - \frac{U}{L} \left(L - \sum_{j=1}^{k_{\max}} t_j \right) = \frac{U}{L} \sum_{j=1}^{k_{\max}} t_j \,.$$

By Lemma 3.1 it holds that

$$\frac{U}{L} \sum_{j=1}^{k_{\max}} t_j = \frac{U}{L} (U - (m-1) t_{k_{\max}}) = \frac{U}{L} s_1$$

due to the choice of s_1 in the configuration $\mathcal{S}_{k_{\max}}$. Thus the makespan would be at least $U/L = \overline{\rho}(m, b)$.

The makespan was at least U/L in both cases, hence the robustness factor is at least $U/L = \overline{\rho}(m, b)$ and the theorem follows. ◀

4 Pebbles

Recall that an instance of our problem is called q-pebbles if the processing times satisfy

$$p_j \leq q \cdot \frac{P}{m} = q \cdot \frac{\sum_{\ell=1}^{n} p_\ell}{m} \,.$$

This definition might seem a bit unnatural at the first glance, but there is a very intuitive formulation. The expression $\frac{P}{m}$ represents the *average load of a machine*. The definition of pebbles says that the processing times are relatively small compared to the average load of all machines.

Without loss of generality we assume in this section that the sum of processing times is $P = m$. This transforms the condition for q-pebbles from the definition into

$$p_j \leq q \,,$$

which is easy to work with.

We use similar ideas as in the optimal algorithm for sand. Recall the condition of Theorem 2.1

$$a_k \leq \frac{\rho P - \sum_{j=1}^{k-1} a_j}{m} \,.$$

As we have already noticed in Section 3.1, the optimal bag sizes for sand not only satisfy the above inequality, they actually have equality there. The bag sizes for sand are given by the recurrence

$$a_k = \frac{\overline{\rho}(m, b) P - \sum_{j=1}^{k-1} a_j}{m} \,.$$

When we in addition assume $P = m$, as in the case of pebbles, we get

$$a_k = \overline{\rho}(m,b) - \frac{1}{m}\sum_{j=1}^{k-1} a_j . \tag{4.1}$$

Let $a_1, \dots, a_b$ denote values given by the recurrence (4.1) for the rest of this section. Remember that the sum of $a_1, \dots, a_b$ equals P. Let us denote the bag sizes we will be choosing for pebbles $d_1, \dots, d_b$. We again want to use Theorem 2.1. In other words, for the desired robustness factor ρ, we want the bag sizes to satisfy

$$d_k \le \rho - \frac{1}{m}\sum_{j=1}^{k-1} d_j . \tag{4.2}$$

Consider the following algorithm. Place as many pebbles as you can into the first bag while it satisfies the inequality (4.2). Then do the same for the second bag and so on until the last bag (or until we run out of jobs). See Pebbles for pseudocode.

Algorithm Pebbles.

Input: processing times $p_1 \ge \cdots \ge p_m$; number of machines m; number of bags b; desired robustness factor ρ
$B \leftarrow$ empty mapping
for $k \leftarrow 1$ to b **do** $d_k \leftarrow 0$ ▷ d_k represents the size of the kth bag
$k \leftarrow 1$ ▷ k represents index of currently considered bag
for $j \leftarrow 1$ to n **do**
  while $k \le b$ **and** $d_k + p_j > \rho - \frac{1}{m}\sum_{\ell=1}^{k-1} d_\ell$ **do** $k \leftarrow k+1$
  if $k > b$ **then break**
  $B[j] \leftarrow k$
  $d_k \leftarrow d_k + p_j$
return B

▶ **Theorem 4.1.** *There exists a $(\overline{\rho}(m,b)+q)$-robust algorithm for q-pebbles, for $\overline{\rho}$ defined by* (1.1).

Proof. We show that Algorithm Pebbles puts every job in some bag for $\rho = \overline{\rho}(m,b) + q$.

Suppose for a contradiction that the algorithm does not use all the jobs. Then the bag sizes d_k at the end of the algorithm must satisfy

$$d_k + q > \rho - \frac{1}{m}\sum_{j=1}^{k-1} d_j .$$

Indeed, if for some k this inequality is not satisfied, adding one more job of size at most p to bag k would not violate the inequality (4.2) and the algorithm would have done so.

Plugging in the expression for ρ gives us

$$d_k > \overline{\rho}(m,b) - \frac{1}{m}\sum_{j=1}^{k-1} d_j . \tag{4.3}$$

We are going to show

$$\sum_{j=1}^{k} d_j \ge \sum_{j=1}^{k} a_j , \tag{4.4}$$

for all $k \in \{0, \dots, b\}$. We prove this claim by induction. The case $k = 0$ is trivial since the summations are empty and both sides are equal to 0. Let us now prove the induction step for k using the equation (4.1) and the inequality (4.3).

$$d_k - a_k \geq \left(\overline{\rho}(m,b) - \frac{1}{m}\sum_{j=1}^{k-1} d_j\right) - \left(\overline{\rho}(m,b) - \frac{1}{m}\sum_{j=1}^{k-1} a_j\right) = -\frac{1}{m}\left(\sum_{j=1}^{k-1} d_j - \sum_{j=1}^{k-1} a_j\right)$$

We can now easily finish the induction step. We simplify

$$\begin{aligned}\sum_{j=1}^{k} d_j - \sum_{j=1}^{k} a_j &= \left(\sum_{j=1}^{k-1} d_j - \sum_{j=1}^{k-1} a_j\right) + (d_k - a_k) \\ &\geq \left(\sum_{j=1}^{k-1} d_j - \sum_{j=1}^{k-1} a_j\right) - \frac{1}{m}\left(\sum_{j=1}^{k-1} d_j - \sum_{j=1}^{k-1} a_j\right) = \frac{m-1}{m}\left(\sum_{j=1}^{k-1} d_j - \sum_{j=1}^{k-1} a_j\right),\end{aligned}$$

which is non-negative by the induction hypothesis for $k - 1$ and thus the claim (4.4) holds.

Using the claim (4.4) for $k = b$ gives us

$$\sum_{j=1}^{b} d_j \geq \sum_{j=1}^{b} a_j = P\,,$$

which is a contradiction with the assumption that we did not use all jobs. ◀

It is interesting to take a look at the case $b = m$. Theorem 4.1 implies that there exists an algorithm with robustness factor at most

$$\frac{e}{e-1} + q \approx 1.58 + q\,.$$

The best know result for rocks gives robustness factor $2 - 1/m$. This gets arbitrarily close to 2 for large m. Hence we have obtained a stronger result for

$$q < 2 - \frac{e}{e-1} \approx 0.42\,.$$

5 Bricks

In this section, we study the case of jobs with equal processing times. An important parameter is the ratio of the number of jobs and the number of machines, which we denote $\lambda = n/m$. We can scale the instance so that $p_j = 1$ for all j, which we assume from now on. Note that now $P = n$ and the average load is $P/m = n/m = \lambda$.

Thus the instance satisfies the definition of p-pebbles for $p = 1/\lambda$. Theorem 4.1 immediately implies our first improved bound for bricks:

▶ **Theorem 5.1.** *There exists an algorithm with robustness factor at most $\overline{\rho}(b,m) + m/n$ solving the problem for n bricks, m machines and b bags.* ◀

In the rest of this section we focus on our main result, the 1.6-robust algorithm for bricks in case $b = m$. This will have the following ingredients:

- For $\lambda \geq 60$ we have $e/(e-1) + 1/60 < 1.6$, so by Theorem 5.1 we can use Algorithm Pebbles.
- For $\lambda < 60$ we design a new algorithm Bricks. We split the analysis into two cases.

- For $m \geq 144$, we modify its solution into a certain fractional solution, which is easier to analyze, and bound the difference between the two solutions.
- For $m < 144$, we have a finite number of instances, which we verify using a computer.

We stress that the analysis of instances for $m < 144$ shows that Algorithm Bricks works here without any changes, too, i.e., it does not lead to an algorithm with exploding number of cases tailored to specific inputs.

5.1 First stage algorithm Bricks

Our assumptions on the optimal solution explained at the beginning of Section 2 imply that we can also restrict ourselves to instances with $\sum_{i=1}^{m} s_i = n$ and furthermore the values of speeds s_i are integral, as in the optimal solution the machine loads are necessarily integral. (Recall that this is due to the fact that we can modify the speeds independently of the first-stage algorithm.)

The key ingredient of the improved algorithm is to observe that the integrality of speeds allows us to use the pigeonhole principle to create larger bags. Furthermore, with appropriate accounting we can use the pigeonhole principle iteratively.

Let us demonstrate this on an example. Let $n = 13$, $m = 10$ and $\rho = 1.6$. The total speed of 10 machines is 13, so one machine has speed at least 2. This means that one of the machines will have capacity $2\rho = 3.2$ and we can create and assign a bag of size $\lfloor 2\rho \rfloor = 3$. Without integrality of the speeds, only a machine with speed 1.3 would be guaranteed, so the capacity would be just above 2.

To continue iteratively, we cannot reason about the capacity as in Algorithm GreedyAssignment. Instead, for each bag we reserve some integral amount of speed on one of the machines. For this accounting, we represent the remaining unreserved total speed by *coins*.

In the example above, we pay 2 coins for a bag of size 3. This seems like an overpayment compared to Algorithm GreedyAssignment, as the 2 coins correspond to capacity 3.2, so we waste a capacity of 0.2. However, after this step, we are left with 11 coins among the 10 machines, and using the integrality and the pigeonhole principle once more, we are guaranteed to have one machine with 2 coins (these coins may be on a different machine or they may be the ones remaining on the same machine). Thus we can create another bag of size 3. Now there are only 9 coins remaining and we can only create a bag of size 1 at cost 1. See Figure 5.1 for an illustration. Overall, the effect of integrality is more significant than the overpayment due to rounding, and thus we are able to obtain an improved algorithm.

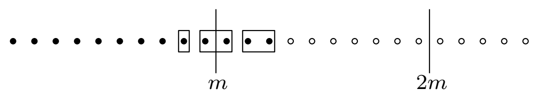

Figure 5.1 Graphical representation of the first three chosen bags for $n = 13$, $m = 10$. The dots represent coins and the boxes represent chosen bags. The number of coins inside a box represent the cost of the bag. Vertical lines emphasize the multiples of m, which determine the bag costs.

Formally, we start Algorithm Bricks with $c = n$ coins. In each round we pay $z = \lceil c/m \rceil$, create a bag of size $\lfloor z \cdot \rho \rfloor$ and continue with remaining coins on m machines. The *cost* of a bag is the number of coins we pay for it, i.e., z in the algorithm.

If Algorithm Bricks produces bags of total size at least n, we say it is *successful*. If the total sum of bag sizes exceeds n, we decrease the sizes of some bags to make the sum equal to n. E.g., we can remove some of the last small bags and then decrease size of the last non-empty bag as needed.

Algorithm BRICKS.

Input: number of bricks n; number of machines m; number of bags b; desired robustness factor ρ

$c \leftarrow n$ ▷ The initial number of coins is n

for $j \leftarrow 1$ to b **do**

  $z \leftarrow \lceil c/m \rceil$ ▷ max guaranteed coins on a machine

  $a_j \leftarrow \lfloor z \cdot \rho \rfloor$ ▷ max integer such that $\text{cost}(a_j) = z$

  $c \leftarrow c - z$

return $a_1, a_2, \ldots, a_b$

In Section 5.3 we show that this algorithm is sound, namely, we give a modification of the second stage algorithm algorithm GREEDYASSIGNMENT for which we show that a machine with unused speed z always exists and thus we can assign all bags.

For a general instance, there is always a machine of speed at least $\lceil n/m \rceil = \lceil \lambda \rceil$, and thus the cost of the first bag is chosen as $\lceil \lambda \rceil$. The cost will then decrease by 1 every time the number of coins decreases below a multiple of m. Figure 5.2 illustrates this.

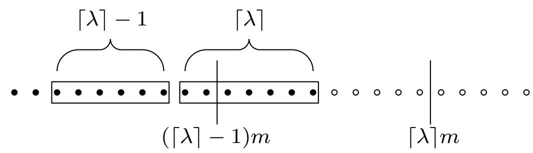

Figure 5.2 Graphical representation of the first chosen bag of size $\lceil \lambda \rceil$.

Note that the costs of the bags chosen by BRICKS do not depend on ρ. The sizes of the bags, however, do depend on ρ. See Figure 5.3 below for an example execution of BRICKS for $n = 45$ and $m = 9$. This execution shows that BRICKS fails for $\rho < 1.6$ but succeeds for $\rho = 1.6$.

5.2 Fractional solutions

In general, the cost of the first bag chosen will be $\lceil \lambda \rceil$. The cost will then decrease by 1 every time the number of coins decreases below a multiple of m. Roughly speaking, we use approximately m coins for bags of each size.

We need to show that the created b bags have total size at least n. If we would use exactly m/z bags for each cost z, the total size of bags is easy to compute. However, the integral number of bags of each cost causes rounding issues when the bag cost decreases and these complicate the calculations.

To structure our analysis, we first modify the solution obtained by BRICKS into a solution that uses possibly non-integral number of bags of each size. In such a solution, we can use fractions of bags (such as $\frac{4}{5}$ of a bag of size 8 as in the Figure 5.4). We arrange the modification so that the total size of bags of cost z is exactly m, except for the smallest and largest bag costs. In the main part of our proof, we bound the rounding error, i.e., the difference between the sizes of the integral and fractional solution. To complete the proof, we calculate the total size of bags in the modified fractional solution, which is easy, and show that it is well above n.

For the fractional solutions, it is better to use an alternative representation of the bags by a function F that for each z gives the number $F(z)$ of bags of cost z. The size of F is then defined as the total size of bags. Recall that a bag of cost z has size $\lfloor z \cdot \rho \rfloor$. Formally:

remaining coins c	a_j for $\rho = 1.6$	a_j for $\rho < 1.6$
45	8	< 8
40	8	< 8
35	6	≤ 6
31	6	≤ 6
27	4	≤ 4
24	4	≤ 4
21	4	≤ 4
18	3	≤ 3
16	3	≤ 3
	46	≤ 44

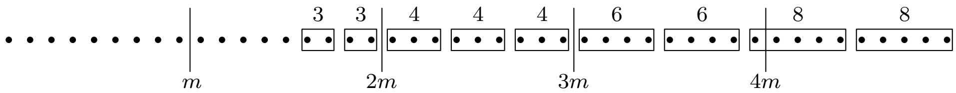

Figure 5.3 Tabular and graphical representation of the execution of BRICKS for $n = 45$, $m = 9$ and $\rho = 1.6$. The numbers above bags represent their sizes. The sum of bag sizes is actually $46 > n = 45$, to solve this, we can for example replace one bag of size 3 with a bag of size 2.

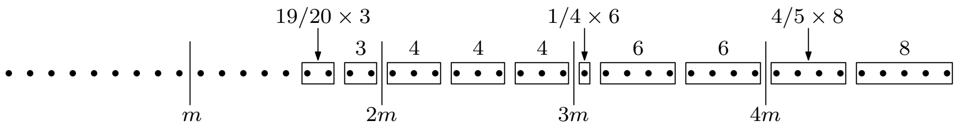

Figure 5.4 Fractional solution for $n = 45$, $m = 9$ and $\rho = 1.6$ produced by BRICKSFRACT. Notice that we always use only one bag size (cost) between consecutive multiples of m. Compare this to Figure 5.3 where bag cost 5 "overflows" the line at $4m$ coins.

▶ **Definition 5.2.** *A fractional solution is a mapping $F : \mathbb{N} \to \mathbb{R}_0^+$ satisfying $\sum_{z=1}^{\infty} F(z) = b$. The size of fractional solution F for robustness factor ρ is defined as*

$$\text{size}(F, \rho) = \sum_{z=1}^{\infty} F(z) \cdot \lfloor z \cdot \rho \rfloor .$$

We will sometimes use only size(F) *if ρ is clear from the context.*

We start by reformulating BRICKS so that it produces the solution directly in the alternative representation, see Algorithm BRICKSALT below. It is easy to see that BRICKS and BRICKSALT are equivalent.

▶ **Observation 5.3.** BRICKS *and* BRICKSALT *use each bag cost the same number of times.*

Proof. One step of BRICKSALT corresponds to several steps of BRICKS. BRICKS chooses the bags one by one, and it may choose the same bag cost in several consecutive iterations. BRICKSALT in each step calculates how many bags of given cost would BRICKS use. The key observation is that the expression $\lceil (c - m(z-1))/z \rceil$ calculates how many bags of cost z are needed to have at most $m(z-1)$ coins remaining. In other words, it calculates how many bags of cost z BRICKS uses before it starts using bags of cost $z-1$ (or runs out of bags). Hence both BRICKS and BRICKSALT use the same number of bags of cost z for each z. ◀

Algorithm BRICKSFRACT (see below) is obtained from BRICKSALT by removing the rounding in the calculation of the number of bags x.

Algorithm BricksAlt.

Input: number of bricks n; number of machines m; number of bags b
Output: Fractional solution I
$r \leftarrow b$ ▷ r is the remaining number of bags (integral)
$c \leftarrow n$ ▷ c is the remaining number of coins (integral)
$I[z] \leftarrow 0$ for $z \in \mathbb{N}$
while $r > 0$ **and** $c > 0$ **do**
  $z \leftarrow \lceil \frac{c}{m} \rceil$ ▷ z is the bag cost
  $x \leftarrow \min\left(r, \lceil \frac{c-m(z-1)}{z} \rceil\right)$ ▷ x is the (integral) number of bags of cost z
  $r \leftarrow r - x$
  $c \leftarrow c - x \cdot z$
  $I[z] \leftarrow x$
return I

Algorithm BricksFract.

Input: number of bricks n; number of machines m; number of bags b
Output: Fractional solution F
$r \leftarrow b$ ▷ r is the remaining number of bags (fractional)
$c \leftarrow n$ ▷ c is the remaining number of coins (fractional)
$F[z] \leftarrow 0$ for $z \in \mathbb{N}$
while $r > 0$ **and** $c > 0$ **do**
  $z \leftarrow \lceil \frac{c}{m} \rceil$
  $x \leftarrow \min\left(r, \frac{c-m(z-1)}{z}\right)$ ▷ x is the fractional amount of bags of cost z
  $r \leftarrow r - x$
  $c \leftarrow c - x \cdot z$
  $F[z] \leftarrow x$
return F

The following observation says that the algorithm follows our initial intuition, namely that for bags of each cost we use exactly m coins, except for the first and last bag cost used.

▶ **Definition 5.4.** *Let F be a fractional solution, then let $z_{\min}$ and $z_{\max}$ denote the smallest and largest integers such that $F(z_{\min}) > 0$ and $F(z_{\max}) > 0$.*

▶ **Observation 5.5.** *Let F be a result* BricksFract *with input n and m. Then $F(z) = m/z$ for every z such that $z_{\min} < z < z_{\max}$.*

Proof. Observe that in every step of the algorithm, except the last one, it holds that $x = (c - m(z-1))/z$ and thus $c - x \cdot z = m(z-1)$. It follows that in all the steps except for the first and last ones $x = (mz - m(z-1))/z = m/z$. ◀

Next we observe that the result of BricksFract scales, i.e., essentially it depends only on λ. Note also that BricksFract is well defined even for non-integral m and n.

▶ **Observation 5.6.** *Let $\alpha \in \mathbb{R}^+$. Suppose* BricksFract *produces solution F with n and m as an input and solution $\bar{F}$ with inputs αn and αm. Then $\bar{F}(z) = \alpha F(z)$ for all z. It follows that $\mathrm{size}(\bar{F}, \rho) = \alpha \cdot \mathrm{size}(F, \rho)$.*

Proof. We go through the execution of BricksFract step by step. Suppose that we multiply both m and n by α. Then in every iteration of the loop r is multiplied by α, c is multiplied by α, z stays the same, and x is multiplied by α. ◀

Now we are ready to bound the difference between the solutions produced by Bricks-Fract and BricksAlt, i.e., the rounding error.

▶ **Lemma 5.7.** *Let F be the fractional solution produced by* BricksFract *and I the solution produced by* BricksAlt *on the same input. Then*

- *for $\lambda \leq 5$ it holds that* $\mathrm{size}(I, 1.6) \geq \mathrm{size}(F, 1.6)$ *and*
- *for $\lambda \leq 60$ it holds that* $\mathrm{size}(I, 1.6) \geq \mathrm{size}(F, 1.6) - 12$*.*

Proof. We give an algorithm which transforms F into a solution F' that is almost integral and very close to I. Set $\bar{z}$ to be z_{min} of the solution F and note that $z_{\mathrm{max}} = \lceil \lambda \rceil$.

We go through the bag costs, denoted by z, from $\lceil \lambda \rceil$ down to $\bar{z} + 1$. For each z, if F uses non-integral amount of bags of cost z, round it up. This makes the number of bags of cost z equal to their number in the solution I. Then decrease the number of bags of cost $z - 1$ so that the total cost of all bags remains the same. Finally, increase the number of bags of cost 1 so that the total number of bags stays equal to b. See Figure 5.5 for an illustration.

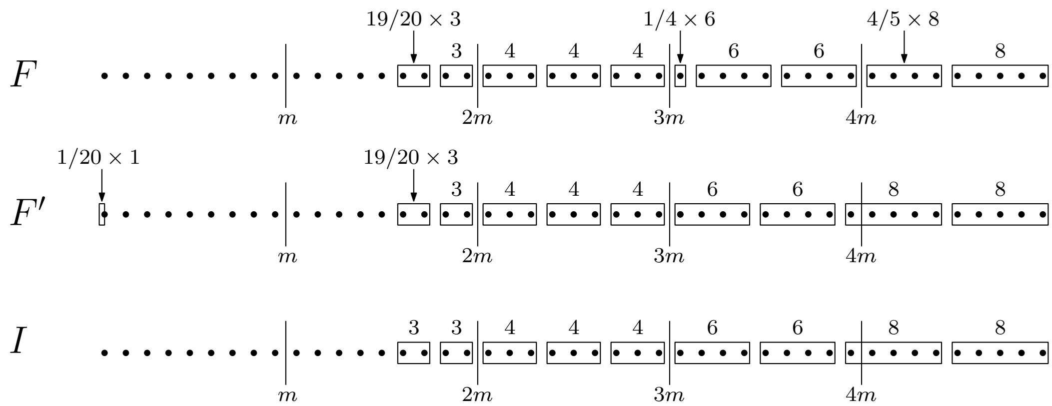

Figure 5.5 Graphical representation of F, F' and I. In the first step of the transformation from F to F', $\frac{9}{5}$ is rounded up to 2 and the number of bags of cost 5 (and size 8) increases by $\frac{1}{5}$. In order to keep the total cost the same, number of bags of cost 4 (and size 6) is decreased by $\frac{1}{4}$. As a result, total number of bags decreased by $\frac{1}{4} - \frac{1}{5} = \frac{1}{20}$, hence we add $\frac{1}{20}$ of a bag of cost 1 (and size 1). This is actually the only step in which something happens since number of used bags of cost 4 and 3 is already integral. We do not process the bags of cost 2, as $\bar{z} = 2$. Solution F' is almost identical to the solution I, but has $\frac{1}{20}$ of bag of cost 1 instead of $\frac{1}{20}$ of bag of cost 2.

For $z = \bar{z} + 1$, the previous procedure could lead to a negative value of $F'(\bar{z})$. In this special case we proceed slightly differently and instead of rounding $G(z)$ up we only increase it so that $F'(\bar{z}) = 0$.

We now describe one step of the process formally and analyze it. Let G denote the current fractional solution and let H denote the result of one transformation step. Let z be the current cost of bags.

We set $H(z) = \lceil G(z) \rceil$, note that $H(z) - G(z) < 1$. We want the sum of costs of bags of costs $z - 1$ and z to remain the same, hence we want

$$H(z) \cdot z + H(z-1) \cdot (z-1) = G(z) \cdot z + G(z-1) \cdot (z-1) \tag{5.1}$$

to hold. Rearranging (5.1) to an equivalent equation leads to (5.2), so we set

$$H(z-1) = G(z-1) + (G(z) - H(z)) \cdot \frac{z}{z-1}\,. \tag{5.2}$$

We claim that $H(z-1) > 0$ for $z-1 > \bar{z}$. Indeed, as $m > 144$ and $z \leq \lceil\lambda\rceil \leq 60$ in the considered case, we have $G(z-1) = F(z-1) = m/(z-1) > 2$ (using also $\bar{z} = z_{\min} < z-1 < z_{\max}$). As $H(z) - G(z) < 1$, we get $H(z-1) > G(z-1) - 1 > 0$.

Now we describe the modification in the special case when $H(z-1)$ would become negative. We have shown above that this can happen only for $z = \bar{z}+1$, i.e., in the last step. Then we set $H(z-1) = 0$ and set

$$H(z) = G(z) + (G(z-1) - H(z-1))\frac{z-1}{z}.$$

This equation is equivalent to (5.1), which is in turn equivalent to (5.2), which thus also holds. Furthermore, the fact that the previous procedure would lead to negative $H(z-1)$ implies that now we have $G(z) \leq H(z) \leq \lceil G(z)\rceil$ and thus $H(z) - G(z) < 1$ holds again.

In both cases, the total number of bags has decreased by

$$(G(z) - H(z)) + (G(z-1) - H(z-1)) = \frac{1}{z-1}(H(z) - G(z))\,.$$

Thus we set

$$H(1) = G(1) + \frac{1}{z-1}(H(z) - G(z))\,.$$

Note that in the transformation step, both the total number of bags and their total cost remain constant.

Recall that the size of a bag of cost z is $\lfloor z\rho\rfloor$. It follows that

$$\begin{aligned}
&\mathrm{size}(H) - \mathrm{size}(G)\\
&\quad = (H(z) - G(z)) \cdot \lfloor z\rho\rfloor + (H(z-1) - G(z-1)) \cdot \lfloor (z-1)\rho\rfloor + (H(1) - G(1)) \cdot \lfloor \rho\rfloor\\
&\quad = (H(z) - G(z)) \cdot \left(\lfloor z\rho\rfloor - \frac{z}{z-1}\lfloor (z-1)\rho\rfloor + \frac{1}{z-1}\lfloor\rho\rfloor\right)
\end{aligned}$$

Note that the second factor in the expression above does not depend on the solution. We call it the transformation factor and for z we denote it by

$$f(z) = \left(\lfloor z\rho\rfloor - \frac{z}{z-1}\lfloor (z-1)\rho\rfloor + \frac{1}{z-1}\lfloor\rho\rfloor\right)\,.$$

If $f(z) \geq 0$, the size of the solution could have only increased, as $H(z) \geq G(z)$, i.e., we have $\mathrm{size}(H) \geq \mathrm{size}(G)$. If $f(z) < 0$, the size of the solution might have decreased – those are the important ("bad") cases we need to bound. We have $H(z) - G(z) < 1$, hence $\mathrm{size}(H) \geq \mathrm{size}(G) + f(z)$ in case of negative $f(z)$.

Now we sum these bounds over all steps for z from $\lceil\lambda\rceil$ to 2 and get

$$\mathrm{size}(F') - \mathrm{size}(F) \geq \sum_{z=2}^{\lceil\lambda\rceil} \min(0, f(z))$$

We give a list of values of $f(z)$ for z from 2 to 60 and $\rho = 1.6$ in the full version of the paper on arXiv [8]. For $z \leq 5$ the values $f(z)$ are non-negative, thus for $\lambda \leq 5$ we get $\mathrm{size}(F') - \mathrm{size}(F) \geq 0$. It can be verified that the sum of all negative values of $f(z)$ for $z \leq 60$ is larger than -12 and thus for $\lambda \leq 60$ we get $\mathrm{size}(F') - \mathrm{size}(F) > -12$.

Examining the algorithms BricksAlt and BricksFract that generate the solutions I and F, respectively, and the transformation process above shows that the solution F is step by step transformed towards I. In particular, $I(z) = F'(z)$ for all values $z \geq \bar{z}$ (if the special case does not apply) or ($z \geq \bar{z} + 1$ if the special case applies). For the small values of z, the only possible difference is that solution F' might have some amount of bags of size 1 instead of some larger bags in solution I. (Note that the total number of bags does not change during the transformation.) This implies size$(I) \geq$ size(F') and the lemma follows. ◀

To complete the proof we need to show that size(F) is sufficiently large so that size$(I) \geq n$. Actually, as the previous transformation possibly gives I with a slightly smaller size than F, we need to compensate for this difference which is at most 12. Precisely, we need to prove that size$(F, 1.6) \geq n$ for $\lambda \leq 5$ and size$(F, 1.6) \geq n + 12$ for $5 < \lambda \leq 60$ and $m \geq 144$.

Since the fractional solution F scales when m and n are scaled, see Observation 5.6, it is convenient to normalize by m and consider (size$(F, 1.6) - n)/m$ in the following lemma. Let us call this crucial quantity *normalized brick surplus*, as it measures how many bricks we are able to put in the bags in the fractional solution in addition to n bricks, normalized by m.

▶ **Lemma 5.8.** *Let F be a fractional solution produced by* BricksFract*. Then*
- *For $\lambda \leq 4$ it holds that* size$(F, 1.6) \geq n$.
- *For $4 \leq \lambda \leq 60$ it holds that* size$(F, 1.6) \geq n + \frac{1}{12}m$.

Proof. By Observation 5.6, the normalized brick surplus size$(F, 1.6) - n)/m$ is uniquely determined by λ, i.e., multiplying both n and m by the same constant does not change it.

This means that the normalized brick surplus is a function of λ. Furthermore, we claim that the function is piece-wise linear. Suppose we slowly increase λ, for example fix m and increase n by δ. Then $F(z)$ remains constant for all z except $z_{\min}$ and $z_{\max}$ by Observation 5.5. The number of the largest bags $F(z_{\max})$ increases by $\delta/\lceil\lambda\rceil$ and $F(z_{\min})$ decreases by the same amount; this amount is proportional to δ. The function size(F) is linear in the values of $F(z)$. So the normalized brick surplus is piece-wise linear with possible breakpoints between the segments at the values of λ when one of the values of $z_{\min}$ or $z_{\max}$ changes.

The value of $z_{\max}$ changes exactly when λ is an integer. The breakpoints where $z_{\min}$ increases can be calculated in the following way: Execute BricksFract for all integer values of $\lambda \leq 60$. Let us denote one of such solutions F. Take a look at $F(z_{\min})$, if we now slowly increase λ, $F(z_{\min})$ will decrease linearly as described above. Calculate at which point it reaches 0; if it happens before λ increases above another integer, we found a point where $z_{\min}$ changes. The first case of changing $z_{\min}$ is at $\lambda = \frac{11}{3}$ when we stop using bags of cost 1, see Figure 5.6.

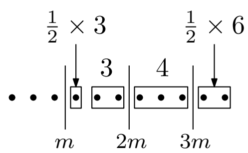

Figure 5.6 Example of solution produced by BricksFract for $n = 11$, $m = 3$ and $\rho = 1.6$. Size of this solution is 11.5 and normalized brick surplus is $\frac{1}{6}$. The solution does not use any bags of cost 1. However, if λ were smaller, the solution would use bags of size 1.

The computer-generated tables of values of the normalized brick surplus function are given in the full version of the paper on arXiv [8]. The plot of the values is given in Figures 5.7 and 5.8 below.

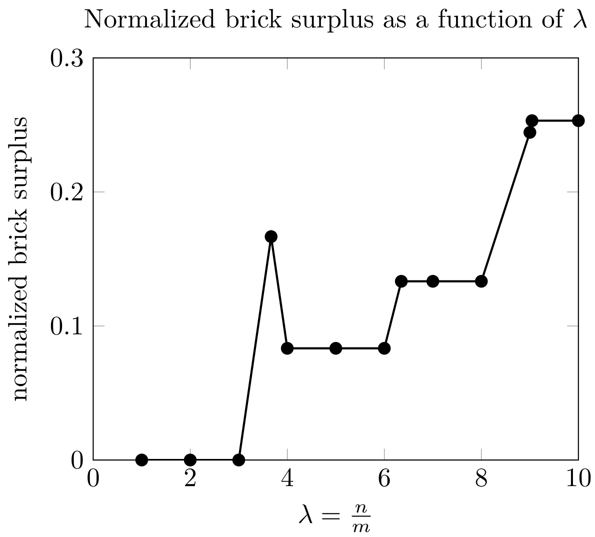

Figure 5.7 Plot of normalized brick surplus for small λ.

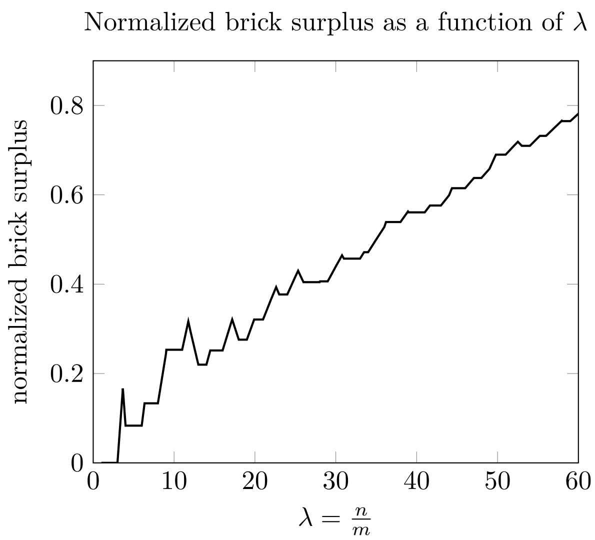

Figure 5.8 Plot of normalized brick surplus for large λ.

The lemma now follows, since the normalized brick surplus is always non-negative and it is at least $1/12$ for $\lambda \geq 5$. (Note that it is a constant function equal to $1/12$ for $\lambda \in [4, 6]$.) ◀

▶ **Theorem 5.9.** *For $\rho = 1.6$ and $\lambda \leq 60$, Algorithm* Bricks *always succeeds, i.e., outputs bags of total size at least n.*

Proof. For $\lambda \leq 5$, the first claims in Lemmata 5.8 and 5.7 together prove $\text{size}(I, 1.6) \geq \text{size}(F, 1.6) \geq n$.

For $5 \leq \lambda \leq 60$ and $m \geq 144$ the second claims in Lemmata 5.8 and 5.7 together prove $\text{size}(I, 1.6) \geq \text{size}(F, 1.6) - 12 \geq n + m/12 - 12 \geq n + 144/12 - 12 = n$.

For $\lambda \leq 60$ and $m \leq 144$ there are only a finitely many instances and we verify $\text{size}(I, 1.6) \geq n$ for them by computer, see the full version of the paper on arXiv [8]. ◀

We note that our choice of the bounds in the previous two lemmata is somewhat arbitrary. The plots of the normalized brick surplus suggest that we could bound it by an appropriate linear function instead of a constant. Also, the bound on $\text{size}(F) - \text{size}(I)$ can be made smaller for intermediate values of λ. These changes would decrease the number of cases we need to check by a computer program, but would not improve the robustness factor.

5.3 Second stage

We need to show that if Bricks succeeds, in the second stage we can indeed achieve makespan ρ. To do this, we cannot use Algorithm GreedyAssignment and Theorem 2.1. Instead we modify it to Algorithm IntegralAssignment below, which copies the coins accounting scheme from Bricks and thus follows the intuition behind it.

Algorithm IntegralAssignment.

Input: bag sizes $a_1 \geq \cdots \geq a_b$; machine speeds $s_1 \geq \cdots \geq s_m$; desired robustness factor ρ
for $i \leftarrow 1$ to m **do**
  $c_i \leftarrow s_i$ ▷ Machine i gets s_i coins at the beginning.
  $M_i = \emptyset$ ▷ Initialize the assignment
for $k \leftarrow 1$ to b **do**
  $i \leftarrow$ index of the machine with the largest c_i
  $M_i \leftarrow M_i \cup \{k\}$ ▷ Assign bag k to machine i
  $c_i \leftarrow c_i - \lceil a_k/\rho \rceil$ ▷ Machine i pays for the bag k
return $M_1, \ldots, M_m$

▶ **Theorem 5.10.** *Suppose the first-stage algorithm* Bricks *succeeds, i.e., outputs bags of total size of at least n. Then* IntegralAssignment *in the second stage produces an assignment with makespan at most ρ.*

Proof. Imagine that Bricks and IntegralAssignment are running in parallel. Bricks chooses the size of one bag and IntegralAssignment assigns it to a machine. Note that the values of c_i remain integral during the entire execution.

We claim that during the execution the value c in Bricks is at most $\sum_{i=1}^{m} c_i$ for c_i's in IntegralAssignment. At the beginning, the quantities are equal. Suppose that Bricks creates a bag of cost z and thus decreases c by z. Then the bag has size $a = \lfloor z \cdot \rho \rfloor \leq z\rho$. Thus IntegralAssignment decreases c_i by $\lceil a/\rho \rceil \leq \lceil z\rho/\rho \rceil = z$. Thus the sum of c_i's decreases by at most z and the claim follows.

The claim implies that in each step before creating/assigning a bag of cost z, there exists a machine with $c_i \geq z$ in IntegralAssignment. Indeed, Bricks chooses $z = \lceil c/m \rceil$, thus $m(z-1) < c \leq \sum_{i=1}^{m} c_i$ using the previous claim. Hence there exists a machine with $c_i > z - 1$ and together with integrality of c_i we get $c_i \geq z$.

It follows that c_i's remain non-negative during the execution. Thus IntegralAssignment assigned to machine i bags of the total size at most $s_i \cdot \rho$. It follows that the makespan is at most ρ. ◀

Theorems 5.1, 5.9, and 5.10 immediately imply our main result.

▶ **Theorem 5.11.** *There exists 1.6-robust algorithm for the case of bricks and $b = m$.* ◀

Conclusions

Our main result still leaves a small gap in the bounds for bricks (equal-length jobs) and $b = m$ between the lower bound of $e/(e-1) \approx 1.58$ and our upper bound of 1.6. Our algorithm Bricks does not admit a smaller robustness factor than 1.6, as is shown for $n = 45$ and $m = 9$ in Figure 5.3. So a smaller upper bound would need some additional techniques or special handling of some cases. Eberle *et al.* [4] give an example that shows a lower bound for bricks that is larger than $\overline{\rho}(m, m)$ for $m = 6$. Although the value of the bound is below the limit value $e/(e-1)$, this may be taken as a weak evidence that matching the lower bound may be hard.

The main open problem in this model remains to find a $(2-\varepsilon)$-robust algorithm for the general case and $b = m$.

References

1 Susanne Albers. Algorithms for dynamic speed scaling. In *Proc. of the 28th Int. Symp. on Theoretical Aspects of Computer Science, STACS 2011*, volume 9 of *LIPIcs*, pages 1–11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPICS.STACS.2011.1.

2 Bo Chen, André van Vliet, and Gerhard J. Woeginger. An optimal algorithm for preemptive on-line scheduling. *Oper. Res. Lett.*, 18(3):127–131, 1995. doi:10.1016/0167-6377(95)00039-9.

3 Florian Diedrich, Klaus Jansen, Ulrich M. Schwarz, and Denis Trystram. A survey on approximation algorithms for scheduling with machine unavailability. In Jürgen Lerner, Dorothea Wagner, and Katharina Anna Zweig, editors, *Algorithmics of Large and Complex Networks – Design, Analysis, and Simulation [DFG priority program 1126]*, volume 5515 of *Lecture Notes in Computer Science*, pages 50–64. Springer, 2009. doi:10.1007/978-3-642-02094-0_3.

4 Franziska Eberle, Ruben Hoeksma, Nicole Megow, Lukas Nölke, Kevin Schewior, and Bertrand Simon. Speed-robust scheduling: sand, bricks, and rocks. *Math. Program.*, 197(2):1009–1048, 2023. A preliminary version appeared at 22nd IPCO, vol 12707 of LNCS, pages 283–296, Springer, 2021. doi:10.1007/S10107-022-01829-0.

5 Leah Epstein, Asaf Levin, Alberto Marchetti-Spaccamela, Nicole Megow, Julián Mestre, Martin Skutella, and Leen Stougie. Universal sequencing on an unreliable machine. *SIAM J. Comput.*, 41(3):565–586, 2012. doi:10.1137/110844210.

6 Dorit S. Hochbaum and David B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM J. Comput.*, 17(3):539–551, 1988. doi:10.1137/0217033.

7 Csanád Imreh and John Noga. Scheduling with machine cost. In *Proc. of the 3rd Int. Workshop on Randomization and Approximation Techniques in Computer Science and 2nd Int. Workshop on Approximation Algorithms for Combinatorial Optimization Problems, RANDOM-APPROX'99*, volume 1671 of *Lecture Notes in Computer Science*, pages 168–176. Springer, 1999. doi:10.1007/978-3-540-48413-4_18.

8 Josef Minařík and Jiřrí Sgall. Speed-robust scheduling revisited. *arXiv e-prints*, 2024. arXiv:2407.11670.

9 Kirk Pruhs. Speed scaling. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms – 2016 Edition*, pages 2045–2047. Springer, 2016. doi:10.1007/978-1-4939-2864-4_390.

10 Kirk Pruhs, Jirí Sgall, and Eric Torng. Online scheduling. In Joseph Y.-T. Leung, editor, *Handbook of Scheduling – Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004. doi:10.1201/9780203489802.CH15.

11 Clifford Stein and Mingxian Zhong. Scheduling when you do not know the number of machines. *ACM Trans. Algorithms*, 16(1):9:1–9:20, 2020. A preliminary version appeared at 29th SODA, pages 1261–1273, ACM, 2018. doi:10.1145/3340320.

12 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. doi:10.1017/CBO9780511921735.