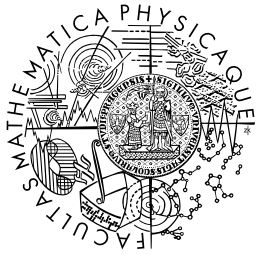

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Josef Minařík

Two-phase scheduling with unknown speeds

Computer Science Institute of Charles University

Supervisor of the master thesis: prof. RNDr. Jiří Sgall, DrSc.

Study programme: Computer Science – Discrete Models and Algorithms

Prague 2024

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank my supervisor prof. RNDr. Jiří Sgall, DrSc. for introducing me to this topic and for his assistance on this thesis.

Title: Two-phase scheduling with unknown speeds

Author: Josef Minařík

Institute: Computer Science Institute of Charles University

Supervisor: prof. RNDr. Jiří Sgall, DrSc., Computer Science Institute of Charles University

Abstract: Speed-robust scheduling is a two-stage scheduling problem with a makespan objective. We are given processing times of n jobs, number of machines m and number of bags b. We have to group the jobs into bags that are to be scheduled on machines of currently unknown speed. The goal is to minimize the worst-case ratio of our makespan and makespan of an adversary who does not have to create bags and assigns jobs directly to machines. So far, the problem has been mostly studied for $b = m$.

We generalize previously known results for infinitesimal jobs (called *sand*) and prove that the best achievable competitive ratio is $\frac{m^b}{m^b-(m-1)^b}$. We present an algorithm for the case of identical jobs (called *bricks*) with competitive ratio at most 1.6 in the case $b = m$, improving the best previously known value of 1.8.

We introduce a new category called *p-pebbles*, those are jobs with processing time at most p times the average load of a machine. Pebbles are half way between sand and the general case (called *rocks*). We present an algorithm for pebbles that has better robustness factor than the best known algorithm for rocks for small values of p (for p less than $2 - \frac{e}{e-1}$ in the case $b = m$).

Keywords: scheduling, approximation algorithms, makespan, uniform speeds

Název práce: Dvoufázové rozvrhování s neznámými rychlostmi

Authr: Josef Minařík

Katedra: Informatický ústav Univerzity Karlovy

Vedoucí diplomové práce: prof. RNDr. Jiří Sgall, DrSc., Informatický ústav Univerzity Karlovy

Abstrakt: Rychlostně robustní rozvrhování je dvoufázový rozvrhovací problém. Na vstupu je dána doba běhu pro každý z n úkolů, počet strojů m a počet balíčků b. Naším úkolem je rozdělit úkoly do balíčků, které budou následně zpracovány na strojích, které mají v tomto okamžiku neznámé rychlosti. Naším cílem je minimalizovat poměr délky našeho rozvrhu a délky optimáního rozvrhu, který by vznil umisťováním úkolů rovnou na stroje. Nejpodrobněji studovaný případ doposud byl $b = m$.

V této práci zobecňujeme známé výsledky pro infinitizemálně malé úkoly (tento případ se nazývá *písek*) a dokážeme, že nejlepší kompetitivní poměr, kterého je možné dosáhnout, je $\frac{m^b}{m^b-(m-1)^b}$. Dále formulujeme algoritmus řešící případ s identickými úkoly (nazývaný *cihly*) za podmínky $b = m$ s kompetitivním poměrem 1.6, což zlepšuje nejlepší doposud známou hodnotu 1.8.

Zavedeme nový speciální případ, který budeme nazývat *p-oblázky*. V tomto případě jsou doby běhu jednotlivých úkolů nejvýše p-násobky průměrné zátěže stroje. Oblázky jsou vlastnostmi i obtížností na půl cesty mezi pískem a obecným případem (nazývaným *kameny*). Popíšeme algoritmus pro oblázky, který je pro malé hodnoty p lepší než nejlepší známý algoritmus pro kameny (pro p menší než $2 - \frac{e}{e-1}$ v případě $b = m$).

Klíčová slova: rozvrhování, aproximační algoritmy, délka rozvrhu, uniformní rychlosti

Contents

1. Introduction

Scheduling is a well known class of optimization problems. We are given a set of jobs, sometimes called tasks or problems, and they are to be scheduled on several machines, also called processors or workers. The goal is to optimize some objective function. Objective functions may vary, but we usually want to finish as many jobs as possible in as little time as possible. Common objective is makespan minimization, where we want to minimize the moment of completion of the last finished job. The exact formulation of objective functions, of course, depends on the problem we are solving. Most scheduling problems are motivated by real-world situations, often by assigning tasks to some computational cluster.

The jobs and machines might have different properties and there can be relations/dependencies between them. For example, processing time of a job might depend or a machine; or some jobs might have other jobs as prerequisites. Many specific settings have been thoroughly studied and there are many known results. One of the main branches of scheduling are online scheduling problems, which usually involve solving common scheduling problems with some kind of uncertainty.

We will study a problem called speed-robust scheduling. Vaguely speaking, speed-robust scheduling is a two stage problem, where we have to group jobs into bags, which are going to be scheduled on machines of unknown speed. The speeds are then revealed, and bags are scheduled to machines with makespan objective. Our makespan is then compared to the makespan of the adversary, who can assign the jobs to machines arbitrarily and does not have to create bags. Our goal is to minimize the worst-case ratio of our makespan and of the makespan of the adversary. A precise definition of speed-robust scheduling, as well as an overview of known results, will be given later in this introduction. Previous work was mostly focused on the case when number of bags equals number of machines. We generalize some of known results for larger number of machines and sometimes prove a stronger statements.

We will first give a brief overview of more common scheduling problems related to speed-robust scheduling.

1.1 Scheduling jobs on parallel machines

Let us briefly introduce a well-known problem of scheduling jobs on parallel machines with a makespan objective.

1.1.1 Identical machines

We are given n jobs and m identical machines. The jobs are to be processed by the machines. Every job ought to be processed by one of the machines; the machines can run in parallel and every machine can be working on at most job at a given time. The jobs have *processing times* $p_1, \ldots, p_n$, i.e. job i takes p_i units of time to process. Our goal is to assign the jobs to machines in a way that minimizes the time it takes to finish all jobs.

More formally, given two integers n, m, and non-negative reals $p_1, \ldots, p_n$, our task is to produce a mapping $M : \{1, \ldots, n\} \to \{1, \ldots, m\}$ representing the assignment of jobs to machines. We define *load* of machine i as

$$L_i = \sum_{j \in M^{-1}(i)} p_j.$$

The *completion time* of such machine, denoted by C_i, is then equal to the load

$$C_i = L_i.$$

The *makespan* of an assignment is

$$C_{\max} = \max_{1 \leq i \leq m} C_i.$$

The objective is to minimize the makespan $C_{\max}$.

It is well known that the problem of scheduling jobs on parallel identical machines is NP-complete. There exists a polynomial time approximation scheme and the existence of a fully polynomial approximation scheme would imply P = NP [1].

1.1.2 Machines with different speeds

One natural generalization of the problem is to allow machines with different speeds. This problem is sometimes called scheduling on uniform machines. We are in addition given non-negative reals $s_1, \ldots, s_m$ representing the speeds of the machines. We allow speeds to be zero but require existence of at least one machine with positive speed. The speeds work in an intuitive way. If a machine has zero speed, no jobs may be scheduled on it. Machines with higher speeds finish jobs faster. Machine i with a non-zero speed and load L_i has a completion time

$$C_i = \frac{L_i}{s_i}.$$

Completion time of machines with zero speed (and thus no assigned jobs) is defined to be equal to 0. This problem can be approximated efficiently as well, i.e. there exists a polynomial time approximation scheme [2].

1.2 Scheduling with incomplete information

There are many ways to introduce some form of uncertainty into scheduling.

One common way to do so is to consider online scheduling where the jobs arrive one by one. The algorithm has to schedule each job on some machine immediately after it arrives and without knowledge of what jobs will follow. We will describe *preemptive* variant of this problem in more detail.

1.2.1 Preemptive online scheduling

In this model, all machines are identical and the jobs arrive one by one. After the arrival of any job, we must assign it on one or more machines and time slots.

The sum of lengths of those time slots must be equal to the processing time of the job, and all the slots must be disjoint. The objective is to minimize the time at which all jobs are finished.

The optimal competitive ratio for preemptive online scheduling is

$$\frac{m^m}{m^m - (m-1)^m}$$

and the optimal solution uses a geometric sequence with ratio $\frac{m-1}{m}$ [3]. We will see a connection between preemptive online scheduling and special case of speed-robust scheduling in Chapter 2.

1.3 Speed-robust scheduling

Speed-robust scheduling is a two-stage problem that was introduced by Eberle *et al.* [4, 5]. Let us introduce speed-robust scheduling on a situation that could occur in the real world.

Suppose that you have n computational tasks that you want to solve. You have a computational cluster available, however you do not know the parameters of the cluster. You only know that there will be (at most) m machines available on the cluster. At the moment, you do not know anything about the performance of the machines—some of the machines might be faster than others. You can, however, submit at most b different tasks to the cluster. Hence you will have to group 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 (with respect to the speeds of the machines) on the machines and minimize the makespan. Some groupings might of course better than others—for example, it is probably not a good idea to put all your tasks into one group. The goal of speed-robust scheduling is to find best possible grouping.

Let us summarize the two stages with an example.

1. In the first stage, the speeds of the machines are unknown. Jobs must be grouped into $b \geq m$ bags (some of which might be empty). We can see an example of this stage in Figure 1.1.

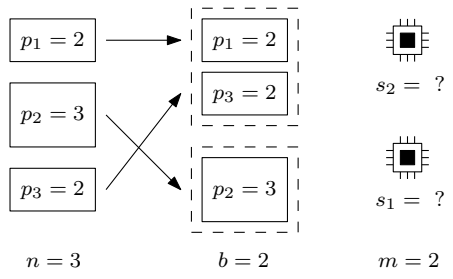

Figure 1.1: An example of the first stage. Two bags of sizes 3 and 4 were created. Note that the speeds of the machines are unknown at this point.

2. In the second stage, speeds of all the machines are revealed. Each bag is assigned to one of the machines, multiple bags can be assigned to one machine. You can see an example of this stage in Figure 1.2.

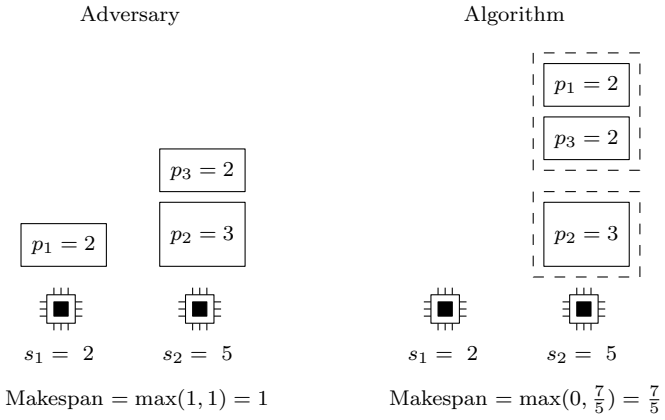

Figure 1.2: An example of the second stage. It is optimal to assign both bags to the second machine to achieve a makespan equal to $\frac{7}{5}$. Note that we are allowed to leave the first machine without any assigned bag. The adversary could achieve makespan equal to 1.

The second stage is just scheduling on parallel machines with different speeds, but instead of assigning jobs, we are assigning bags. We assume that the second stage is solved optimally, and our goal is to solve the first stage. To evaluate the performance of our algorithms, we consider the worst-case ratio of the algorithm's makespan and the makespan of the adversary. The adversary does not create bags and assigns jobs directly to machines of known speeds. In other words, the adversary is solving an offline scheduling on machines with different speeds.

The formulation of speed-robust scheduling used by Eberle *et al.* is slightly different [5]. In their paper, they use the following alternative formulation: The adversary also creates bags, but the adversary already knows the speeds of all machines at the time of bag creation (in the first stage). It is easy to see that our formulation is equivalent for $b \geq m$.

In parts of this text, we only consider the special case $b = m$. There are very few previously known results for general b, with the notable exception of Theorem 5.1.

1.3.1 Formal definition and notation

Formally, in the first stage, we receive three positive integers n, m, b such that $b \geq m$ and n non-negative real numbers $p_1, \ldots, p_n$. We will denote $P = \sum_{i=1}^{n} p_i$. The output of our 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.

Let us call the sum of the processing times of all the jobs assigned to bag i

the *size* of bag i. Formally, the size of the bag i is

$$a_i = \sum_{j:B(j)=i} p_j.$$

The exact mapping B is not important for the second stage since the makespan depends only on the bag sizes. See FirstStageTemplate, it is a template of a first-stage algorithm.

Algorithm FirstStageTemplate

Input: Number of jobs n
Number of machines m
Number of bags b
Processing times of the jobs $p_1, \ldots, p_n$

Output: Grouping of jobs into bags - mapping $B : \{1, \ldots, n\} \to \{1, \ldots, b\}$.

In the second stage, the speeds $s_1, s_2, \ldots, s_m$ are revealed and at least one of them is non-zero. Let $C_1, \ldots, C_m$ denote completion times of machines in the optimal solution of the second stage and let $C_{\max}$ be the makespan.

Let $C_1^*, \ldots, C_m^*$ denote completion times of machines in the solution of the adversary and let $C_{\max}^*$ be the makespan.

We are interested in the ratio

$$\frac{C_{\max}}{C_{\max}^*}.$$

Note that it depends on the speeds $s_1, \ldots, s_m$ which are unknown in the first stage.

We will call an algorithm ρ-*robust* if

$$\frac{C_{\max}}{C_{\max}^*} \leq \rho$$

holds for all possible inputs and for all possible choices of machine speeds. An algorithm is ρ-robust if it always performs at most ρ times worse than the adversary. The *robustness factor* of an algorithm is defined as an infimum over all such ρ.

Note that the case $n \leq b$ is somewhat trivial because we can achieve robustness factor 1 by placing every job into a separate bag.

1.3.2 Terminology

We will call the special cases of the problem *sand*, *bricks*, *rocks* and *pebbles*. Sand, bricks, and pebbles were introduced by Eberle *et al.* [5]. These words represent the types of jobs we will be dealing with.

- 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 the same processing times.

- Sand grains are very small and represent infinitesimally short processing times.
- Pebbles represent jobs that are relatively small compared to the average load of all machines. See Chapter 3 for formal definition.

Formally, we will need to use a slightly different formulation of the first stage for sand, see Chapter 2 for details.

1.3.3 Variants

A special case of speed-robust scheduling has been studied by Steiner and Zhong [6]. They considered a variant of the problem where $s_i \in \{0, 1\}$ for $1 \leq i \leq m$. In other words, they assumed that there are m machines, but some of them might fail and not be able to process any jobs. They presented a $\frac{5}{3}$-robust algorithm for rocks and proved that there is no algorithm with a robustness factor lower than $\frac{4}{3}$.

1.4 Assumptions

In the rest of this text, we will make the following assumptions, unless explicitly stated differently.

- Processing times, machine speeds and bag sizes are sorted in a descending order.
- The makespan of the adversary is 1. Multiplying all the speeds by a positive real constant clearly does not change the ratio of the makespans of our algorithm and the adversary. Formally, for every instance of the problem, there exists another instance where $C^*_{\max} = 1$ and the ratio of makespans of our algorithm and the adversary is the same.
- The sum of the processing times of all jobs equals to the sum of the speeds of all the machines, i.e.
$$\sum_{i=1}^{m} p_i = \sum_{i=1}^{m} s_i.$$
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. We can make this assumption because we are investigating the worst case and all the other cases are "easier". If there is some machine i with $s_i > 0$ and $C^*_i < 1$, we can consider
$$s'_i = C^*_i s_i,$$
which does not change the makespan of the adversary (which is equal to 1 by the first assumption), and this change can only increase our makespan.

1.5 Results

Eberle *et al.* [5] have shown that the best achievable robustness factor for sand in the case of $b = m$ is

$$\overline{\rho}(m) = \frac{m^m}{m^m - (m-1)^m}.$$

We generalize this result for $b \geq m$ and prove that the best achievable robustness factor for b bags and m machines is

$$\overline{\rho}(m, b) = \frac{m^b}{m^b - (m-1)^b}$$

in Chapter 2. See Theorem 2.2.

We introduce a new special case in Chapter 3. We consider instances of the problem called p-pebbles, where

$$p_i \leq p \cdot \frac{P}{m}$$

holds for all jobs i. This guarantees that there are no *big* jobs and allows us to show the existence of an $(\overline{\rho}(b, m) + p)$-robust algorithm. For small p, this gives better bound than the currently strongest known result for rocks.

Eberle *et al.* [5] presented an algorithm for bricks in case $b = m$ with a robustness factor of at most

$$\left(1 + \frac{m}{n}\right) \overline{\rho}(m)$$

for any m and n. They have also presented an 1.8-robust algorithm for any n and m; again in the case $b = m$. We strengthen both results in Chapter 4 and generalise the first one for $b \geq m$. We use results for pebbles and present an algorithm with robustness factor at most

$$\overline{\rho}(m, b) + \frac{m}{n}$$

for any n, $b \geq m$ (Theorem 4.1). For the special case of $b = m$, we give an 1.6-robust algorithm for any n and m (Theorem 4.6). We use mathematical arguments for large m and n and computer search for small instances.

The strongest known result for rocks is the existence of an algorithm with robustness factor at most $1 + \frac{m-1}{b}$ [5]. See Chapter 5 and Theorem 5.1.

1.6 General ideas of the proofs

Many of the proofs of robustness will have one thing in common: Although we are allowed to solve the second stage optimally, we do not. The reason for that is quite simple, the optimal solution of the second stage can be complicated, and thus it is difficult to bound its makespan. We will instead provide a simple (usually greedy) algorithm for the second stage which will be much easier to work with. See SecondStageTemplate for a template of a second-stage algorithm.

In particular, GreedyAssignment will be very useful. It is additionally parameterized by ρ, which represents the robustness factor that we want to achieve.

Algorithm SecondStageTemplate

Input: Bag sizes $a_1 \geq \cdots \geq a_b$
Machine speeds $s_1 \geq \cdots \geq s_m$

Output: The mapping of bags to machines $M : \{1, \ldots, b\} \to \{1, \ldots, m\}$.

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 and assigns them on the machine with the most capacity remaining. The assignment of a job of size a decreases the capacity of the selected machine by a.

Algorithm GreedyAssignment

Input: Bag sizes $a_1 \geq \cdots \geq a_b$
Machine speeds $s_1, \ldots, s_m$
Desired robustness factor ρ

for $j \leftarrow 1$ to m **do**
  $c_j \leftarrow \rho s_j$ ▷ Initialize the capacities of all machines
end for
$M \leftarrow$ empty mapping
for $i \leftarrow 1$ to b **do**
  $j \leftarrow$ index of machine with the largest capacity left
  $M[i] \leftarrow j$ ▷ Assign bag i to machine j
  $c_j \leftarrow c_j - a_i$ ▷ Decrease the remaining capacity of the selected machine
end for
return M

If the capacities remain non-negative at the end of execution of GreedyAssignment, the makespan of the created assignment must clearly be at most ρ since machine j has been assigned jobs of total processing times at most ρs_j. Let us formulate a simple theorem which will be very useful throughout the whole text.

Theorem 1.1. *Suppose that bag sizes $a_1, \ldots, a_b$ satisfy inequalities*

$$a_i \leq \frac{\rho P - \sum_{j=1}^{i-1} a_j}{m}.$$

Then GreedyAssignment *produces an assignment with makespan at most ρ.*

Proof. Recall that we assume $\sum_{i=1}^{m} s_i = P$. We only need to show that there is a machine with capacity at least a_i when assigning the ith bag. This will guarantee that the capacities stay non-negative until the end. The initial total capacity was ρP and was already decreased by $\sum_{j=1}^{i-1} a_j$ at the time of assigning bag a_i. It follows that the average remaining capacity is equal to

$$\frac{\rho P - \sum_{j=1}^{i-1} a_j}{m}.$$

and thus there exists a machine with at least

$$\frac{\rho P - \sum_{j=1}^{i-1} a_j}{m}$$

capacity remaining. □

Corollary 1.2. *If a first-stage algorithm always produces bag sizes satisfying inequalities*

$$a_i \leq \frac{\rho P - \sum_{j=1}^{i-1} a_j}{m},$$

it is ρ-robust.

Proof. This is a direct consequence of Theorem 1.1. We are assuming that makespan of the adversary is 1. According to Theorem 1.1 there exists an assignment with makespan at most ρ for every output of the algorithm, hence it is ρ-robust. □

2. Sand

In this chapter, we consider a special case of the problem. We assume that n is very large and all the jobs are of the same size and very small. You can think of it as of infinite number of jobs and 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.

Intuition behind this case is quite simple. We are given a pile of sand (of volume P) and, in the first stage, we create several smaller piles and put them into bags. The sand is continuous and we can split it into parts of arbitrary sizes.

The formulation of the second stage remains the same.

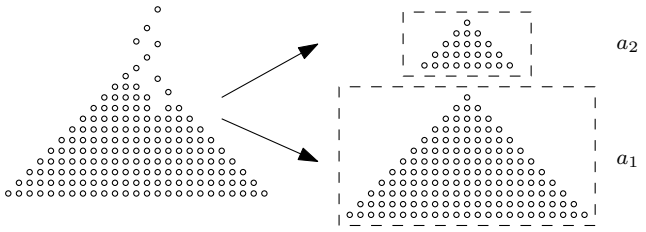

Figure 2.1: Sand can be distributed arbitrarily.

Definition 2.1. *Let* $\overline{\rho} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ *be defined as*

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

In addition, let $\overline{\rho}(m) = \overline{\rho}(m, m)$.

Eberle *et al.* solved the case $b = m$ and showed that the best achievable robustness factor is $\overline{\rho}(m)$ [5]. We generalize this result for $b \geq m$ and significantly simplify the proof of the upper bound.

Theorem 2.2 (Sand)**.** *The best achievable robustness factor for infinitesimal jobs,* m *machines and* b *bags is* $\overline{\rho}(m, b)$.

Before proving this result, let us make some observations about $\overline{\rho}(m, b)$. One may wonder what happens when we are allowed to use much more bags than there are machines. It is easy to show that the optimal robustness factor goes to 1 as b goes to infinity.

Observation 2.3. *For every* $m \in \mathbb{N}$ *it holds that*

$$\lim_{b \to \infty} \overline{\rho}(b, m) = 1.$$

Proof. By simple manipulation of the expression

$$\lim_{b \to \infty} \overline{\rho}(b, m) = \lim_{b \to \infty} \frac{m^b}{m^b - (m-1)^b} = \lim_{b \to \infty} \frac{1}{1 - \left(\frac{m-1}{m}\right)^b} = 1.$$

□

Another natural question is what happens if b is some constant multiple of m. Does the optimal robustness factor also go to 1 as m and b go to infinity if, say, $b = 2m$? The answer is no, the optimal robustness factor converges to $\frac{e^\alpha}{e^\alpha - 1}$ for $b = \alpha m$.

Observation 2.4. *For every $\alpha \in \mathbb{R}^+$ and $m \in \mathbb{N}$ it holds that*

$$\overline{\rho}(\alpha m, m) < \frac{e^\alpha}{e^\alpha - 1}.$$

Furthermore, for every $\alpha \in \mathbb{R}^+$ it holds that

$$\lim_{m\to\infty} \overline{\rho}(\alpha m, m) = \frac{e^\alpha}{e^\alpha - 1}.$$

Proof. Let us manipulate the expression

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

Since $1 + x \leq e^x$ holds for $x \in \mathbb{R}$, we can bound

$$\overline{\rho}(\alpha m, m) \leq \frac{1}{1 - e^{-\alpha}} = \frac{e^\alpha}{e^\alpha - 1}.$$

Similarly

$$\lim_{m\to\infty} \overline{\rho}(\alpha m, m) = \lim_{m\to\infty} \frac{1}{1 - \left(1 - \frac{1}{m}\right)^{\alpha m}} = \frac{1}{1 - e^{-\alpha}} = \frac{e^\alpha}{e^\alpha - 1}.$$

□

One notable corollary of Observation 2.4 is that

$$\overline{\rho}(m) < \frac{e}{e-1} \approx 1.58$$

holds for all $m \in \mathbb{N}$.

We will now prove several lemmata that will be useful in the proof of Theorem 2.2. Let us denote $U = m^b$, $L = m^b - (m-1)^b$ and $t_i = m^{b-i}(m-1)^{i-1}$ for $i \in \{1, \ldots, b\}$. Observe that Theorem 2.2 says $\overline{\rho}(b, m) = \frac{U}{L}$.

Lemma 2.5. *It holds that*

$$\sum_{i=1}^{k} t_i = U - (m-1)t_k.$$

Proof. We will proceed by induction. 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

$$\sum_{i=1}^{k+1} t_i = U - (m-1)t_k + t_{k+1} = U - m^{b-k}(m-1)^k + m^{b-k-1}(m-1)^k = U - (m-1)t_{k+1}.$$

which completes the induction step. □

Note that plugging $k = b$ into Lemma 2.5 gives us that the sum of t_i is L.

To get some intuition behind the algorithm for sand, it might be useful to consider the case $m = 2$. Suppose that $P = L = 2^b - 1$ and let us choose bag sizes $a_i = t_i = 2^{b-i}$. Since everything is a power of two in this case, it is easier to think about. It should be easy to see that we can achieve the robustness ratio of $1 + \frac{1}{2^b-1}$. The adversary can choose any speeds s_1, s_2 such that $s_1 + s_2 = 2^b - 1$. The capacities of the machines (as in GreedyAssignment) will then satisfy $c_1 + c_2 = 2^b$ and thus $\lfloor c_1 \rfloor + \lfloor c_2 \rfloor \geq 2^b - 1$. We can simply express $\lfloor c_1 \rfloor$ in binary and assign the corresponding bags on the first machine.

2.1 Upper bound

We use a different approach than Eberle *et al.* for the proof of the upper bound. We choose the same bag sizes but use different arguments to show that such an algorithm is $\overline{\rho}(b, m)$-robust. We will make use of Theorem 1.1. We will use Sand to choose the bag sizes. Note that the sum of bag sizes produced by Sand is P.

Algorithm Sand

Input: Number of bags b
Number of machines m
Total amount of sand P

$L \leftarrow m^b - (m-1)^b$
for $i \leftarrow 1$ to b **do**
  $a_i \leftarrow t_i \frac{P}{L}$
end for
return $a_1, a_2, \ldots, a_b$

Lemma 2.6 (Sand upper bound)**.** Sand *is* $\overline{\rho}(b, m)$*-robust.*

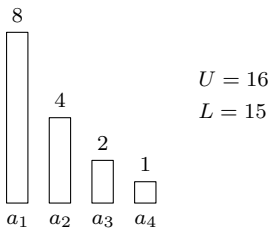

Figure 2.2: An example of bag sizes chosen for $m = 2$ and $b = 4$. If there are only two machines, the bag sizes are powers of two.

Proof. We will 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 $a_i = t_i$.

It is sufficient to show that the bag sizes produced by SAND satisfy the condition of Corollary 1.2. Let us prove the ith inequality.

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

According to Lemma 2.5, we can simplify

$$U - \sum_{j=1}^{i} t_j + t_i = U - (U - (m-1)t_i) + t_i = mt_i = ma_i.$$

This allows us to use Corollary 1.2 and there exists an assignment with makespan at most $\overline{\rho}(b,m)$. □

Notice that the condition of Corollary 1.2 was tight, i.e. there was an equality.

2.2 Lower bound

The following proof is a slightly modified and generalized version of the proof by Eberle *et al.* [5]. The main difference is that we do not require the number of bags and machines to be the same.

Lemma 2.7 (Sand lower bound)**.** *No deterministic algorithm may have a robustness factor smaller than* $\overline{\rho}(b,m)$*.*

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

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

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

$$s_1 = U - (m-1)t_k = \sum_{i=1}^{k} t_i \geq t_k.$$

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

$$\sum_{i=1}^{b} a_i = U = \frac{U}{L}L = \frac{U}{L}\sum_{i=1}^{b} t_i.$$

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

- At least one of the bags $a_1, \ldots, a_{k_{\max}}$ was assigned to a slow machine. The makespan of this machine is at least

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

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

$$U - \sum_{i=1}^{k_{\max}} a_i = U - \sum_{i=k_{\max}+1}^{b} a_i.$$

By definition of $k_{\max}$ it holds that $a_i < \frac{U}{L} t_i$ for $i > k_{\max}$ and we can bound

$$\sum_{i=k_{\max}+1}^{b} a_i \geq U - \frac{U}{L} \sum_{i=k_{\max}+1}^{b} t_i.$$

Since $\sum_{i=1}^{b} t_i = L$, we can rearrange the right-hand side

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

By Lemma 2.5 it holds that

$$\frac{U}{L} \sum_{i=1}^{k_{\max}} t_i = \frac{U}{L} \left(U - (m-1) t_{k_{\max}} \right) = \frac{U}{L} s_1$$

and the makespan would be at least $\frac{U}{L}$.

The makespan was at least $\frac{U}{L}$ in both cases, hence the algorithm cannot be better than $\frac{U}{L}$-robust.

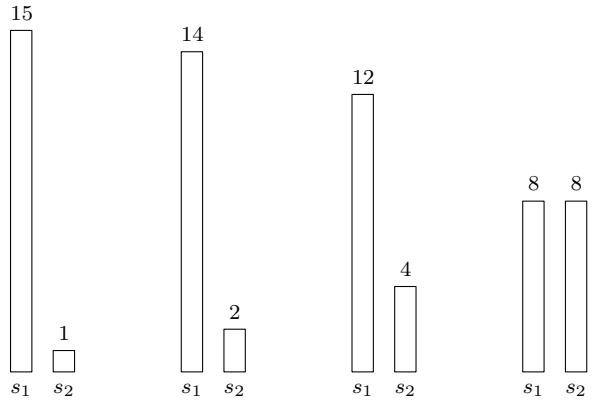

Figure 2.3: An example of speed configurations considered by the adversary for $m = 2$ and $b = 4$.

□

We can now finally prove Theorem 2.2.

Proof of Theorem 2.2. Lemma 2.6 and Lemma 2.7 together prove Theorem 2.2. □

3. Pebbles

It seems to be difficult to use the ideas from sand in the general case of rocks. Since rocks can have arbitrary sizes, it is not really possible to make the bag sizes similar to the ones that would be created by Sand. For this reason, we considered another special case which we call *pebbles*. Pebbles are in between sand and rocks in terms of properties and difficulty. Unlike sand and bricks, pebbles don't assume that all the jobs are the same. They, however, forbid the existence of *large* jobs, which is allowed in rocks.

Definition 3.1. *We will call an instance of the speed-robust scheduling p-pebbles if the processing times satisfy*

$$p_i \leq p \cdot \frac{\sum_{j=1}^{n} p_j}{m} = p \cdot \frac{P}{m}.$$

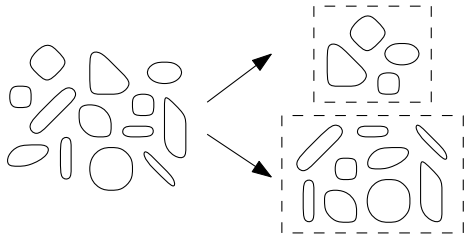

Figure 3.1: Pebbles are relatively small compared to average load of a machine.

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. We will also assume without loss of generality the sum of processing times is m. This will transform the condition from the definition into

$$p_i \leq p,$$

which is easy to work with.

We will use similar ideas as in the optimal algorithm for sand. Recall the condition of Corollary 1.2

$$a_i \leq \frac{\rho P - \sum_{j=1}^{i-1} a_j}{m}.$$

As we have already noticed in Chapter 2, 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

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

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

$$a_i = \overline{\rho}(m, b) - \frac{1}{m} \sum_{j=1}^{i-1} a_j. \tag{3.1}$$

Let $a_1, \ldots, a_b$ denote values given by the recurrence 3.1 for the rest of this chapter. Remember that the sum of $a_1, \ldots, a_b$ equals P. Let us denote the bag sizes we will be choosing for pebbles $d_1, \ldots, d_b$. We will again want to use Corollary 1.2. In other words, we will want the bag sizes to satisfy

$$d_i \leq \rho - \frac{1}{m} \sum_{j=1}^{i-1} d_j, \tag{3.2}$$

where ρ is the desired robustness factor.

Consider the following algorithm. Place as many pebbles as you can into the first bag while it satisfies the inequality (3.2). Then do the same thing 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 $p_1 \geq \cdots \geq p_m$
Number of machines m
Number of bags b
Desired robustness factor ρ

$B \leftarrow$ empty mapping
for $i \leftarrow 1$ to b **do**
  $d_i \leftarrow 0$ ▷ d_i will represent size of the ith bag
end for
$k \leftarrow 1$ ▷ k represents index of currently considered bag
for $i \leftarrow 1$ to n **do**
  while $k \leq b$ **and** $d_k + p_i > \rho - \frac{1}{m} \sum_{j=1}^{k-1} d_j$ **do**
    $k \leftarrow k + 1$
  end while
  if $k > b$ **then**
    break
  end if
  $B[i] \leftarrow k$
  $d_k \leftarrow d_k + p_i$
end for
return B

Lemma 3.2. *The* PEBBLES *puts every job in some bag for* $\rho = \overline{\rho}(m, b) + p$.

Proof. Suppose that the algorithm does not use all the jobs. The bag sizes created by the algorithm must satisfy

$$d_i + p > \rho - \frac{1}{m} \sum_{j=1}^{i-1} d_j.$$

If some bag did not satisfy this inequality, adding one more job would not violate the inequality (3.2) and the algorithm would have done so. Plugging in the expression for ρ gives us

$$d_i > \overline{\rho}(m,b) - \frac{1}{m}\sum_{j=1}^{i-1} d_j. \tag{3.3}$$

We will show

$$\sum_{i=1}^{k} d_i \geq \sum_{i=1}^{k} a_i,$$

for all $k \in \{0,\ldots,b\}$. We will 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 using the equation (3.1) and the inequality (3.3).

$$\begin{aligned} d_k - a_k &\geq \left(\overline{\rho}(m,b) - \frac{1}{m}\sum_{i=1}^{k-1} d_i\right) - \left(\overline{\rho}(m,b) - \frac{1}{m}\sum_{i=1}^{k-1} a_i\right) \\ &= -\frac{1}{m}\left(\sum_{i=1}^{k-1} d_i - \sum_{i=1}^{k-1} a_i\right) \end{aligned}$$

We can now easily finish the induction step. We simplify

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

which is non-negative by the induction hypothesis and thus

$$\sum_{i=1}^{k} d_i \geq \sum_{i=1}^{k} a_i.$$

Setting $k = b$ gives us

$$\sum_{i=1}^{b} d_i \geq \sum_{i=1}^{b} a_i = m,$$

which is a contradiction with the fact that we did not use all jobs. □

Theorem 3.3. *There exists an algorithm for p-pebbles with robustness factor at most* $\overline{\rho}(m,b) + p$.

Proof. PEBBLES places all jobs into bags by Lemma 3.2 and has robustness factor at most $\overline{\rho}(m,b) + p$ because the created bag sizes satisfy the inequality (3.2). □

It is interesting to take a look at the case $b = m$. Theorem 3.3 and Observation 2.4 say that there exists an algorithm with robustness factor at most

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

The best know result for rocks gives robustness factor $2 - \frac{1}{m}$ (Theorem 5.1). This gets arbitrarily close to 2 for large m. Hence we have obtained a stronger result for

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

4. Bricks

In this chapter, we are not given a pile of sand or pebbles but a bunch of bricks instead. The bricks are perfectly regular and all of them are the same size. More formally, we assume $p_i = 1$, that is, all processing times are set to 1.

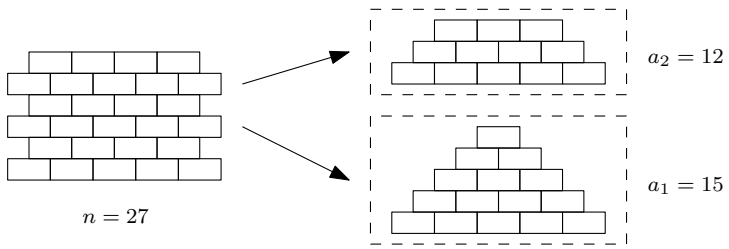

Figure 4.1: All the jobs have processing time 1.

Eberle *et al.* have shown that there exists an algorithm with robustness factor at most 1.8 for $b = m$ [5]. We improve this bound and present an algorithm with robustness factor 1.6. This is much closer to the lower bound of $\frac{e}{e-1} \approx 1.582$ obtained from sand.

Eberle *et al.* have also proven that there exists an algorithm with a robustness factor of at most $(1 + \frac{m}{n})\overline{\rho}(m)$ for any n and m, again under the constraint of $b = m$ [5]. We also strengthen this result and prove the existence of an algorithm with a robustness factor at most $\overline{\rho}(m) + \frac{m}{n}$. We generalize this result for $b \geq m$ and give an algorithm with a robustness factor at most $\overline{\rho}(m, b) + \frac{m}{n}$.

4.1 Bricks are just large pebbles

We can apply the already proven result for pebbles.

Theorem 4.1. *There exists an algorithm with robustness factor at most* $\overline{\rho}(b, m) + \frac{m}{n}$ *solving the problem for* n *bricks,* m *machines and* b *bags.*

Proof. We will use Theorem 3.3. The average load of a machine is $\frac{n}{m}$ and all jobs have size 1. Therefore, bricks are just $\frac{m}{n}$-pebbles and Pebbles is $(\overline{\rho}(b, m) + \frac{m}{n})$-robust for bricks. □

4.2 Using integrality

Let us proceed to description of the algorithm with robustness factor equal to 1.6. We will solve just the special case of $b = m$, i.e., the number of bags equals the number of machines. In the previous section, we did not use the fact that the machine loads selected by the adversary must be integers.

Since we assume that the makespan achieved by the adversary is 1 and the adversary fully utilizes all the machines, the speeds of all machines have to be integers. Otherwise, if the speeds were not integers, we could just round them

down to the nearest integer—this would not change the makespan of the adversary and could only increase our makespan. This assumption implies

$$\sum_{i=1}^{m} s_i = n.$$

Let us denote the ratio $\frac{n}{m}$ by λ, such notaion will be convenient later in this chapter.

Definition 4.2. *We will denote*

$$\lambda = \frac{n}{m}.$$

We could try to use the same greedy algorithm as before. However, if we use GreedyAssignment for second stage, it is quite difficult to make use of the integral speeds in the algorithm analysis. For this reason, we will use a more discrete approach. Instead of assigning a real *capacity* to every machine, we will give each machine a certain number of *coins* initially equal to its speed. We will then *pay* some integral number of coins every time we assign a bag to a machine. *Cost* of a bag is the number of coins we have to pay to assign it to some machine. The cost of a bag will depend on its size and the robustness factor ρ that we want to achieve.

Definition 4.3. *The cost of a bag of size a is*

$$\text{cost}(a, \rho) = \left\lceil \frac{a}{\rho} \right\rceil,$$

where ρ is the robustness factor we want to achieve. We will often write just $\text{cost}(a)$ *since ρ is always fixed in given context.*

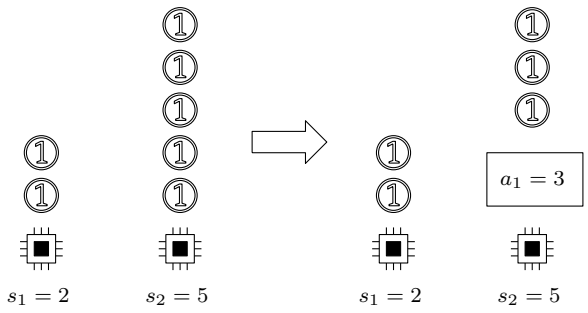

Figure 4.2: Every machine has assigned coins. For example with $\rho = 1.6$, the second machine could pay 2 coins for a bag of size 3.

We can now formulate a modified greedy algorithm for the second stage; see IntegralAssignment. It is very similar to GreedyAssignment, the only difference is that it uses coins instead of capacities.

The number of coins that any machine has during the entire execution of IntegralAssignment remains integral. This will be useful since we will be

Algorithm IntegralAssignment

Input: Bag sizes $a_1 \geq \cdots \geq a_b$
Machine speeds $s_1, \ldots, s_m$
Desired robustness factor ρ

for $j \leftarrow 1$ to m **do**
  $c_j \leftarrow s_j$ ▷ Machine j gets s_j coins at the beginning.
end for
$M \leftarrow$ empty mapping
for $i \leftarrow 1$ to b **do**
  $j \leftarrow$ index of the machine with most coins left
  $M[i] = j$ ▷ Assign bag i to machine j
  $c_j \leftarrow c_j - \text{cost}(a_i)$ ▷ Machine j pays for the job i
end for

able to use the pigeon-hole principle when arguing the existence of a machine with a sufficient number of coins remaining. In particular, if machine j has c_j coins and

$$\sum_{j=1}^{m} c_j > m(\text{cost}(a_i) - 1),$$

then there is at least one machine with $\text{cost}(a_i)$ or more coins left.

Let us demonstrate this on an example. Let $n = 13$, $m = 10$ and $\rho = 1.6$. There are 13 coins distributed between 10 machines, which means that there is at least one machine with at least 2 coins (with speed at least 2). This means that we can assign a bag of size 3 and cost $\lceil \frac{3}{1.6} \rceil = 2$ on one of the machines. Note that we needed the integrality of the speeds, otherwise there would only be guaranteed a machine with speed 1.3. There are 11 remaining coins and we can place another bag of size 3 using the same argument. It is important that all machines have integral number of coins remaining and we can use the pigeon-hole principle again. Now there are only 9 coins remaining and we can only choose a bag of size 1. See Figure 4.3 for an illustration of this situation or Figure 4.6 for more complex example.

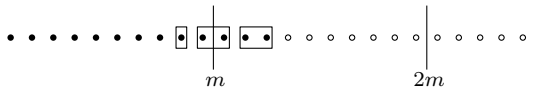

Figure 4.3: Graphical representation of the first three chosen bags for $n = 13$, $m = 10$. The dots represent coins and the boxes represent chosen bags. The number of coins inside a box represent the cost of the bag. Vertical lines emphasize the multiples of m, which determine the bag costs.

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 crosses a multiple of m. Figure 4.4 shows this in a graphical way.

We can observe that if the numbers of coins remain non-negative during the execution of the Algorithm IntegralAssignment, the makespan will be at

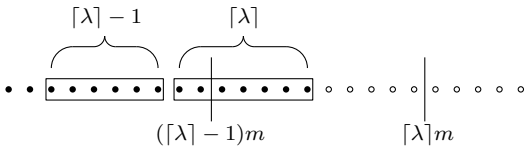

Figure 4.4: Graphical representation of the first chosen bag of size $\lceil\lambda\rceil$.

most ρ. This is true since the sum of bag sizes assigned to a machine i never exceeds ρs_i. IntegralAssignment is actually in some sense *stricter* than Algorithm GreedyAssignment because there is a ceiling function in the definition of cost.

4.3 Choosing the bag sizes

We will now try to choose the bag sizes greedily in a way that guarantees that all machines end up with non-negative number of coins after the execution of the IntegralAssignment. In every step, we will choose the largest possible bag size that is guaranteed to be possible to assign without getting to negative numbers. The exact way of choosing the bag sizes is described by Bricks.

Algorithm Bricks

Input: Number of bricks n
Number of machines m
Desired robustness factor ρ

$c \leftarrow n$ ▷ The initial number of coins is n
for $i \leftarrow 1$ to b **do**
  $k \leftarrow \lceil \frac{c}{m} \rceil$ ▷ max integer such that $c > m(k-1)$
  $a_i \leftarrow \lfloor k \cdot \rho \rfloor$ ▷ max integer such that $\text{cost}(a_i) = k$
  $c \leftarrow c - k$
end for
return $a_1, a_2, \ldots, a_b$

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 4.5 and 4.6 for an example execution of Bricks.

If Bricks produces bags of total size at least n, it was successful. If the total sum of bag sizes exceeds n, we can just decrease the sizes of some bags to make the sum equal to n. We can for example make some of the last bags empty and then decrease size of the last non-empty bag.

Definition 4.4. *We say that* Bricks succeeds *if it produces bags of total size at least n, otherwise we say that the algorithm* fails.

Lemma 4.5. *Suppose the first-stage algorithm* Bricks *succeeds. Then* IntegralAssignment *in the second stage produces an assignment with makespan at most ρ.*

remaining coins c	a_i for $\rho = 1.6$	a_i 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 4.5: Table describing the execution of BRICKS for $n = 45$ and $m = 9$. This table shows that the algorithm fails for $\rho < 1.6$ but succeeds for $\rho = 1.6$.

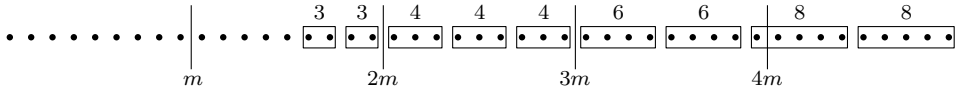

Figure 4.6: Graphical representation of the execution of BRICKS for $n = 45$, $m = 9$ and $\rho = 1.6$. The numbers above bags represent their size. The sum of bag sizes is actually $46 > n = 45$, to solve this, we could for example replace one bag of size 3 with a bag of size 2.

Proof. Imagine that BRICKS and INTEGRALASSIGNMENT are running in parallel. BRICKS chooses the size of one bag and INTEGRALASSIGNMENT assigns it to a machine. During the execution it holds that c from BRICKS equals the sum of c_j from INTEGRALASSIGNMENT. We can verify that the cost of the chosen bag of size a_i is indeed k. It holds that

$$a_i = \lfloor k \cdot \rho \rfloor \leq k \cdot \rho$$

and thus

$$\text{cost}(a_i) = \left\lceil \frac{a_i}{\rho} \right\rceil \leq \left\lceil \frac{k \cdot \rho}{\rho} \right\rceil = \lceil k \rceil = k$$

Similarly it holds that

$$a_i = \lfloor k \cdot \rho \rfloor > k \cdot \rho - 1$$

and thus

$$\text{cost}(a_i) = \left\lceil \frac{a_i}{\rho} \right\rceil \geq \left\lceil \frac{k \cdot \rho - 1}{\rho} \right\rceil = \left\lceil k - \frac{1}{\rho} \right\rceil,$$

which is at least k for $\rho > 1$.

Now we can argue that some of c_j was at least k and thus it remained non-negative. We want to prove that the cost k chosen in the ith step of the BRICKS satisfies

$$c > m(k-1).$$

Since

$$k = \left\lceil \frac{c}{m} \right\rceil < \frac{c}{m} + 1,$$

it holds that

$$m(k-1) < m\left(\frac{c}{m} + 1 - 1\right) = m \cdot \frac{c}{m} = c.$$

□

Theorem 4.6. *There exists 1.6-robust algorithm for the case of bricks for any m and n.*

We have already shown that Pebbles is 1.6-robust for $\lambda \geq 60$ since

$$\frac{e}{e-1} + \frac{1}{60} < 1.6.$$

Thus showing that Bricks always succeeds for $\rho = 1.6$ and $\lambda \leq 60$ will prove Theorem 4.6. Note that Bricks sometimes fails for $\rho < 1.6$. The counterexample for any $\rho < 1.6$ is $n = 45$ and $m = 9$, as shown in Figure 4.5. This is the smallest counterexample in terms of n. For all smaller n it is sufficient to set $\rho = 1.5$.

There are only finitely many cases remaining for a fixed m. We will prove Theorem 4.6 separately for small m and large m separately. We simply try all the cases for small m and use a more involved argument for larger m.

Lemma 4.7. Bricks *always succeeds for $\rho = 1.6$, $\lambda \leq 60$ and $m \leq 144$.*

Proof. Just try all the cases; see Appendix A. □

Lemma 4.8. Bricks *always succeeds for $\rho = 1.6$, $\lambda \leq 60$ and $m > 144$.*

We will prove Lemma 4.8 in the next section. Let us now give the proof of Theorem 4.6.

Proof of Theorem 4.6. Lemma 4.7 and Lemma 4.8 together show that Bricks always succeeds for $\lambda \leq 60$ and thus it is 1.6-robust for $\lambda \leq 60$. There exists 1.6-robust algorithm for $\lambda > 60$ according to Theorem 4.1 since

$$\overline{\rho}(m,m) + \frac{m}{n} = \overline{\rho}(m,m) + \frac{1}{\lambda} < \frac{e}{e-1} + \frac{1}{60} < 1.6.$$

□

4.4 Fractional solutions

To prove Lemma 4.8, we will need to bound the total size of bags produced by Bricks. To do so, we will consider *fractional solutions*. 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 4.7). This does not have a clear combinatorial meaning, but it is quite easy to calculate total bag size of one such fractional solution and then show that the solution produced by Bricks is not much worse.

Algorithm GreedyFractional

Input: Number of bricks n
Number of machines m

Output: Fractional solution F

$r \leftarrow m$ ▷ r is the (real) number of bags remaining
$c \leftarrow n$ ▷ c does not have to remain integral this time
$F[k] \leftarrow 0$ for $k \in \mathbb{N}$
while $r > 0$ **and** $c > 0$ **do**
  $k \leftarrow \lceil \frac{c}{m} \rceil$ ▷ k is the chosen bag cost
  $x \leftarrow \min\left(r, \frac{c-m(k-1)}{k}\right)$ ▷ x is chosen amount of bags of cost k
  $r \leftarrow r - x$
  $c \leftarrow c - x \cdot k$
  $F[k] \leftarrow x$
end while
return F

Definition 4.9. *A fractional solution is a mapping $F : \mathbb{N} \to \mathbb{R}_0^+$ satisfying*

$$\sum_{k=1}^{\infty} F(k) = b.$$

Value $F(k)$ denotes how many bags of cost k the solution uses.

GreedyFractional produces one particular fractional solution. Note that GreedyFractional is well defined even for non-integral m and n (this might be useful for some intuition later). GreedyFractional is, in some sense, very similar to the solution produced by Bricks, it only differs in rounding. Let us formulate Bricks in the language of fractional solutions to highlight the similarity, see IntegralFractional. We can observe that Bricks and IntegralFractional are, in some sense, equivalent.

Observation 4.10. Bricks *and* IntegralFractional *use each bag cost the same number of times.*

Proof. Let us look at the execution of both algorithms. One step of IntegralFractional 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. IntegralFractional instead in each step calculates how many bags of given cost would Bricks use. The key observation is that the expression

$$\left\lceil \frac{c - m(k-1)}{k} \right\rceil$$

calculates how many bags of cost k we must use to have at most $m(k-1)$ coins remaining. In other words, it calculates how many bags of cost k Bricks uses before it starts using bags of cost $k-1$ (unless it runs out of bags). Hence both Bricks and IntegralFractional use the same number of bags of cost k. □

Algorithm IntegralFractional

Input: Number of bricks n
Number of machines m

Output: Fractional solution I

$r \leftarrow m$ ▷ r is the (integral) number of bags remaining
$c \leftarrow n$ ▷ c is integral this time
$I[k] \leftarrow 0$ for $k \in \mathbb{N}$
while $r > 0$ **and** $c > 0$ **do**
  $k \leftarrow \lceil \frac{c}{m} \rceil$
  $x \leftarrow \min\left(r, \lceil \frac{c-m(k-1)}{k} \rceil\right)$ ▷ This is the only difference
  $r \leftarrow r - x$
  $c \leftarrow c - x \cdot k$
  $I[k] \leftarrow x$
end while
return I

Let us now define *size* of a fractional solution, it simply says how many (fractional) bricks are contained in this solution. Remember that a bag of cost k has size $\lfloor k \cdot \rho \rfloor$.

Definition 4.11. *Size of fractional solution F for robustness factor ρ is defined as*

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

We will sometimes use only $\text{size}(F)$ *if ρ is clear from the context.*

Let us now make some observations about the result of GreedyFractional. They will help us to understand the algorithm and will be useful later.

Definition 4.12. *Let F be a fractional solution, then let $k_{\min}$ and $k_{\max}$ denote the smallest and largest integers such that $F(k_{\min}) > 0$ and $F(k_{\max}) > 0$.*

Observation 4.13. *Let F be a result of running* GreedyFractional *with input n and m. Then for every k such that $k_{\min} < k < k_{\max}$ it holds that*

$$F(k) = \frac{m}{k}.$$

Proof. Observe that in every step of the algorithm, except the last one, it holds that

$$x = \frac{c - m(k-1)}{k}$$

and thus

$$c - x \cdot k = m(k-1).$$

Hence c is equal to a multiple of m except in the first step. In all the steps except the first and last it holds that

$$x = \frac{mk - m(k-1)}{k} = \frac{m}{k}.$$

□

Observation 4.14. *Let $\alpha \in \mathbb{R}^+$. Suppose* GreedyFractional *produces solution F with n and m as an input and solution F' with inputs αn and αm. Then it holds that*

$$F'(k) = \alpha F(k)$$

for all k. It follows that

$$\text{size}(F', \rho) = \alpha \cdot \text{size}(F, \rho).$$

Proof. This is very easy to see if you go through the execution of GreedyFractional step by step. Formally, we could show it by induction on the iteration of the while loop. Suppose that we multiply both m and n by α. Then the following claims hold in every iteration of the loop:

- r is multiplied by α,
- c is multiplied by α,
- k stays the same,
- x is multiplied by α.

All of this holds in the first iteration and the induction step is trivial. □

Our goal is to prove

$$\text{size}(I, 1.6) \geq n,$$

where I is solution produced by IntegralFractional, for $m > 144$ and $\lambda \leq 60$. This is equivalent to Lemma 4.8.

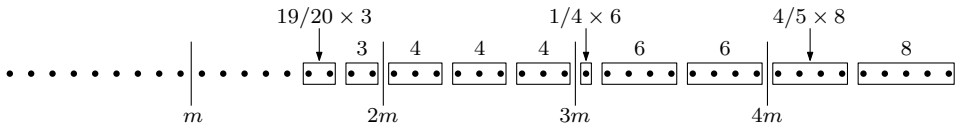

Figure 4.7: Fractional solution for $n = 45$, $m = 9$ and $\rho = 1.6$ produced by GreedyFractional. Notice that we always use only one bag size (cost) between consecutive multiples of m. Compare this to Figure 4.6 where bag of size 8 "oversteps" the $4m$ line.

We shall prove two claims regarding sizes of fractional solutions produced by GreedyFractional and IntegralFractional. The first claim says that the result of GreedyFractional is big and the second says that the result of IntegralFractional is not much smaller.

Claim 4.15. *Let F be a fractional solution produced by* GreedyFractional.

- *For $\lambda \leq 4$ it holds that*

$$\text{size}(F, 1.6) \geq n.$$

- *For* $4 \leq \lambda \leq 60$ *it holds that*

$$\text{size}(F, 1.6) \geq n + \frac{1}{12}m.$$

Claim 4.16. *Let* F *be a fractional solution produced by* GreedyFractional *and let* I *be fractional solution produced by* IntegralFractional*.*

- *For* $\lambda \leq 4$ *it holds that*

$$\text{size}(I, 1.6) \geq \text{size}(F, 1.6).$$

- *For* $4 \leq \lambda \leq 60$ *it holds that*

$$\text{size}(I, 1.6) \geq \text{size}(F, 1.6) - 12.$$

These two claims together allow us to prove Lemma 4.8.

Proof of Lemma 4.8. Claim 4.15 and Claim 4.16 together prove

$$\text{size}(I, 1.6) \geq n$$

for $\lambda \leq 4$ and

$$\text{size}(I, 1.6) \geq n + \frac{1}{12}m - 12 \geq n + \frac{144}{12} - 12 = n$$

for $4 \leq \lambda \leq 60$. □

It remains to prove Claim 4.15 and Claim 4.16.

Proof of Claim 4.15. Consider the expression

$$\frac{\text{size}(F, 1.6) - n}{m}$$

and let us call it *relative brick surplus*. Notice that by Observation 4.14, the relative brick surplus is uniquely determined by λ, i.e., multiplying both n and m by a real positive constant does not change it.

This means that the relative brick surplus is a function of λ. Furthermore, it is piecewise linear. If we slowly increase λ (you can imagine m fixed and increasing n), $F(k)$ remains constant for all k except $k_{\min}$ and $k_{\max}$ by Observation 4.13. The slope only changes when $k_{\min}$ or $k_{\max}$ changes. This only happens when λ is an integer or when we stop using some type of bag when λ is large enough. For example, we stop using bags of cost 1 when λ surpasses $\frac{11}{3}$, see Figure 4.8. This means that we only need to evaluate the relative surplus of bricks in finitely many points. See Appendix B for details. □

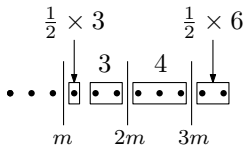

Figure 4.8: Example of solution produced by GreedyFractional for $n = 11$, $m = 3$ and $\rho = 1.6$. Size of this solution is 11.5 and relative brick surplus is $\frac{1}{6}$. The solution does not use any bags of cost 1. However, if λ were smaller (you can imagine $n = 11 - \varepsilon$), the solution would use bags of size 1. If we now slowly increase n, the number of used bags of size 6 will linearly increase while the number of used bags of size 3 will linearly decrease.

Proof of Claim 4.16. We will present an algorithm which transforms the fractional solution F into the solution F' such that

$$\text{size}(I) \geq \text{size}(F') \geq \text{size}(F) - 12.$$

The transformation process is quite simple: go through the bag costs, denoted by k, from $k_{\max} = \lceil \lambda \rceil$ to $k_{\min}$. If F uses non-integral amount of bags of cost k, round it up. Decrease the number of bags of cost $k-1$ so that the total cost of all bags remains the same. Increase the number of bags of cost 1 such that the total number of bags stays equal to b.

Let us analyze one step of the process. Let G denote the current fractional solution and let H denote the result of one transformation step. Let k be the current cost. We set

$$H(k) = \lceil G(k) \rceil.$$

We want the sum of costs of bags of sizes $k-1$ and k to remain the same, hence we want

$$H(k) \cdot k + H(k-1) \cdot (k-1) = G(k) \cdot k + G(k-1) \cdot (k-1)$$

to hold. Thus we set

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

The total number of bags has decreased by

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

Thus we set

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

Remember that the size of a bag of cost k is $\lfloor k \cdot \rho \rfloor$. It follows that

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

Let us denote

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

If $f(k) > 0$, the size could only increase. If $f(k) = 0$, the size remains the same. If $f(k) < 0$, the size might have decreased—those are important ("bad") cases. We can bound

$$0 \leq H(k) - G(k) = \lceil G(k) \rceil - G(k) < 1,$$

hence

$$\text{size}(H) - \text{size}(G) \geq \min\left(0, f(k)\right).$$

Now we just need to sum this difference from $\lceil \lambda \rceil$ to 2 to bound $\text{size}(F') - \text{size}(F)$.

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

It only remains to plug in $\rho = 1.6$ and make the calculation for $\lceil \lambda \rceil = 4$ and $\lceil \lambda \rceil = 60$. We give a list of values of $f(k)$ for k from 2 to 60 in Appendix C. Plugging in $\lceil \lambda \rceil = 4$ gives us

$$\text{size}(F') - \text{size}(F) \geq 0$$

and plugging in $\lceil \lambda \rceil = 60$ gives us

$$\text{size}(F') - \text{size}(F) > -12.$$

Solutions I and F' are almost identical, they may differ only in the smallest bags—solution F' might have some bags of size 1 instead of some larger bags in solution I. This implies

$$\text{size}(I) \geq \text{size}(F') \geq \text{size}(F) - 12.$$

□

See Figure 4.9 for an illustration of the proof of Claim 4.16.

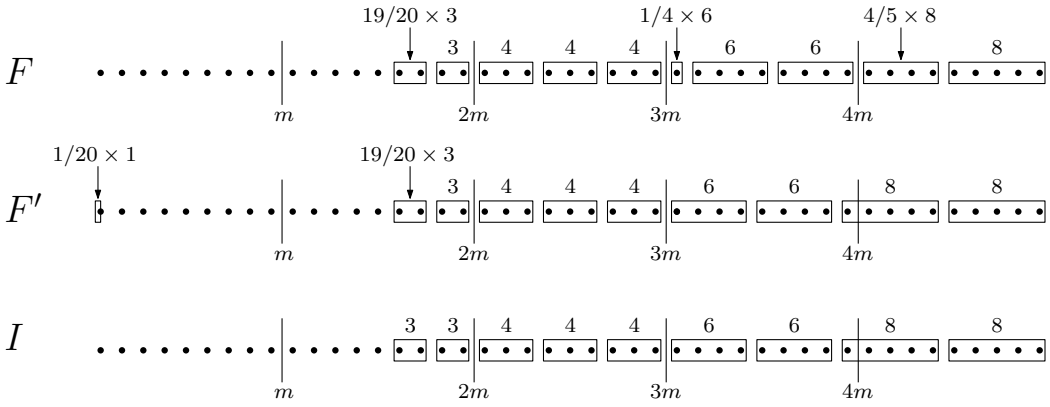

Figure 4.9: Graphical representation of F, F' and I. In the first step of the transformation from F to F', $\frac{9}{5}$ is rounded up to 2 and the number of used bags of cost 5 (and size 8) increases by $\frac{1}{5}$. In order to keep the total cost the same, number of bags of cost 4 (and size 6) decreases by $\frac{1}{4}$. The total amount 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. 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.

5. Rocks

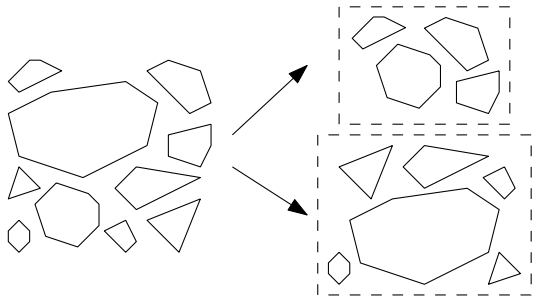

Figure 5.1: Rocks have arbitrary sizes.

In this section, we present a slightly modified version of proof by Eberle *et al.* [5]. We will use *Longest processing time first* algorithm. It takes the jobs in order of decreasing processing time and assigns them to the smallest bag. See Rocks for pseudocode.

Algorithm Rocks

Input: Processing times $p_1 \geq \cdots \geq p_m$
Number of bags b

$B \leftarrow$ empty mapping
for $i \leftarrow 1$ to n **do**
  $j \leftarrow$ index of the bag with smallest size
  $B[i] \leftarrow j$
end for
return B

Theorem 5.1. Rocks *has a robustness factor* $1 + \frac{m-1}{b}$ *for* $b \geq m$.

Proof. We will use a modified version of GreedyAssignment for the second stage. We will partially copy the adversary and then use a greedy algorithm. Note that we cannot copy the adversary completely since they might have created bags different from ours. We assign bags in order from the largest to the smallest. The algorithm starts with a *copycat* phase and switches to a *greedy* phase once it encounters a bag with at least two jobs. In the copycat phase, it simply assigns the bag to the machine on which the adversary placed the only job in the bag. In the greedy phase, it tries to place the bag on a machine with enough capacity left. See Copycat for pseudocode.

The bags assigned in the copycat phase were certainly fine at the time of their assignment since we have placed them on the same machines as the adversary and nothing else there. We need to show that all the capacities remain non-negative in the greedy phase. If there are no bags with at least two jobs, we are done. Now, let $a_{\max}$ be the size of the largest bag containing at least two jobs. Furthermore, let $a_{\min}$ denote the size of the smallest bag; by definition $a_{\min} = a_b$.

Lemma 5.2. *It holds that* $a_{\max} \leq 2a_{\min}$.

Proof. Consider the time when we added the last job to the bag of size $a_{\max}$, let us denote processing time of this job p_{last}. Clearly $p_{\text{last}} \leq \frac{1}{2}a_{\max}$ since $a_{\max}$ contains at least two jobs and p_{last} is the smallest. Because we are using ROCKS for job assignment, it must hold

$$a_{\max} - p_{\text{last}} \leq a_{\min}$$

and Lemma 5.2 follows. □

Suppose we have already placed $k < b$ bags and we are in the greedy phase. We will show that there is a machine with sufficient capacity left for the bag a_{k+1}. We will make use of the Lemma 5.2 and the trivial inequality $a_{k+1} \leq a_{\max}$.

The $b - k$ bags which are not yet assigned have a total size of at least

$$P_{\text{remaining}} \geq (b-k-1)a_{\min} + a_{k+1} \geq \frac{1}{2}(b-k+1)a_{k+1}.$$

The total size of the already assigned bags is at least

$$P_{\text{assigned}} \geq ka_{k+1}.$$

Since the total initial capacity was $\sum_{i=1}^{m} s_i = \sum_{i=1}^{n} p_i = \rho(P_{\text{remaining}} + P_{\text{assigned}})$, the total remaining capacity is at least

$$C \geq \rho P_{\text{remaining}} + (\rho-1)P_{\text{assigned}} = (\rho-1)(P_{\text{remaining}} + P_{\text{assigned}}) + P_{\text{remaining}} \geq$$

$$\geq \frac{1}{2}(\rho-1)(b+k+1)a_{k+1} + \frac{1}{2}(b-k+1)a_{k+1} =$$

$$= \frac{1}{2}\left(\frac{(m-1)(b+k+1)}{b} + b - k + 1\right)a_{k+1}.$$

The expression

$$\frac{(m-1)(b+k+1)}{b} + b - k + 1$$

is decreasing in k since k has linear coefficient

$$\frac{m-1}{b} - 1 < 0.$$

Thus we only need to check the maximum value of k. Let $k = b - 1$, we get

$$C \geq \frac{1}{2}\left(\frac{(m-1)\cdot 2b}{b} + 2\right)a_{k+1} = ma_{k+1}.$$

Thus it follows that there is a machine with capacity at least a_{k+1} which completes the proof. □

Algorithm COPYCAT

Input: Bag sizes $a_1 \geq \cdots \geq a_b$
Machine speeds $s_1, \ldots, s_m$
Desired robustness factor ρ
Assignment used by the adversary

for $j \leftarrow 1$ to m **do**
  $c_j \leftarrow \rho s_j$
end for
$M \leftarrow$ empty mapping
copycat_phase $\leftarrow$ **True**
for $i \leftarrow 1$ to b **do**
  if copycat_phase **and** bag i contains exactly one job **then**
    $k \leftarrow$ index of the job in the bag i.
    $j \leftarrow$ the machine on which was the job k assigned by the adversary.
  else
    copycat_phase $\leftarrow$ **False**
    $j \leftarrow$ index of a machine largest capacity left
  end if
  $M[i] \leftarrow j$
  $c_j \leftarrow c_j - a_i$
end for
return M

Conclusions

We have studied the problem of speed-robust scheduling and we have strenghtened and generalized some of the known results.

We have shown that the best achievable robustness factor for sand in the case of m machines and b bags is

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

We have used a combination of known techniques and new ideas to prove this.

We have introduced a new category called *p-pebbles*. We have applied the results about sand to pebbles and we have presented an algorithm with robustness factor at most

$$\overline{\rho}(m,b) + p.$$

This gives a better result than the best known algorithm for rocks for small values of p.

From the result about pebbles follows existance of an algorithm for bricks with robustness ratio at most

$$\overline{\rho}(m,b) + \frac{m}{n},$$

which improves the previously known result

$$\overline{\rho}(m,b) \cdot \left(1 + \frac{m}{n}\right)$$

and generalizes it for $b \geq m$. We have significantly improved the best known algorithm for bricks for the case $b = m$ achieving robustness factor of 1.6. The question of optimal robustness factor for bricks remains open. However, we have reduced the range of possible values of the best achievable robustness factor for $b = m$ to only

$$\left[\frac{e}{e-1}, 1.6\right].$$

The general case of rocks remains open. The strongest currently known result is the existance of an algorithm with robustness factor at most

$$1 + \frac{m-1}{b}.$$

Bibliography

[1] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[2] 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.

[3] 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.

[4] Franziska Eberle. Scheduling and packing under uncertainty. In Norbert Trautmann and Mario Gnägi, editors, *Operations Research Proceedings 2021, Selected Papers of the International Conference of the Swiss, German and Austrian Operations Research Societies (SVOR/ASRO, GOR e.V., ÖGOR), University of Bern, Switzerland, August 31 - September 3, 2021*, Lecture Notes in Operations Research, pages 9–14. Springer, 2021.

[5] 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.

[6] Clifford Stein and Mingxian Zhong. Scheduling when you don't know the number of machines. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1261–1273. SIAM, 2018.

A. Small cases for bricks

We provide a proof of Lemma 4.7 here. We prove it by trying all possible combinations of m and n such that $m \leq 144$ and $\frac{n}{m} \leq 60$. We provide an implementation in `Python 3.11`. We represent numbers as fractions to avoid issues with numerical precision. We implement Algorithm IntegralFractional, since it is faster than Algorithm Bricks. The code should take at most a few minutes to run on modern hardware.

```
from fractions import Fraction
from math import ceil, floor


def integral_fractional(
    n: int,
    m: int,
) -> dict[int, Fraction]:
    """
    n: number of jobs
    m: number of machines

    returns: Solution in format {bag_cost: number_of_bags}
    """
    c = n
    r = m
    I = {}
    while r > 0 and c > 0:
        k = ceil(Fraction(c, m))
        x = min(r, ceil(Fraction(c - m * (k - 1), k)))
        c -= x * k
        r -= x
        I[k] = x

    return I


rho = Fraction(8, 5)
for m in range(1, 145):
    for n in range(1, 60 * m + 1):
        I = integral_fractional(n, m)
        solution_size = sum(
            floor(bag_cost * rho) * bag_count
            for bag_cost, bag_count in I.items()
        )
        if solution_size < n:
            raise Exception(
                f"Failed for: {n=} {m=} {rho=} \n Solution: {I}"
            )
```

B. Brick Surplus

We will again make use of a computer to handle more difficult calculations. We evaluate the relative brick surplus at all integer values of λ from 1 to 60. In addition, we evaluate for every value of λ when we stop using bags of some cost, i.e. $k_{\min}$ increases. The first case of this happening is $\lambda = \frac{11}{3}$ when we stop using bags of cost 1. We also calculate the relative brick surplus for those non-integral values of λ.

Checking only those values is sufficient, since the brick surplus is linear between the calculated points. See Figure B.3 for values of relative brick surplus at integer values of λ and Figure B.4 for values at non-integer values of λ. See Figures B.1 and B.2 to get a better idea of how the relative brick suplus behaves for small and large values of λ. Note that the relative brick surplus sometimes decreases.

The values of λ for which $k_{\min}$ increases were calculated in the following way: Execute GreedyFractional for all integer values of $\lambda \leq 60$. Let us denote one of such solutions F. Take a look at $F(k_{\min})$, if we now slowly increase λ, $F(k_{\min})$ will linearily decrease. We can calculate at which point it reaches 0; if it happens before λ crosses another integer, we found a point where $k_{\min}$ changes.

Claim 4.15 follows, since the relative brick surplus is always non-negative and is at least $\frac{1}{12}$ for $\lambda \geq 4$. It is equal to $\frac{1}{12}$ for values of λ between 4 and 6.

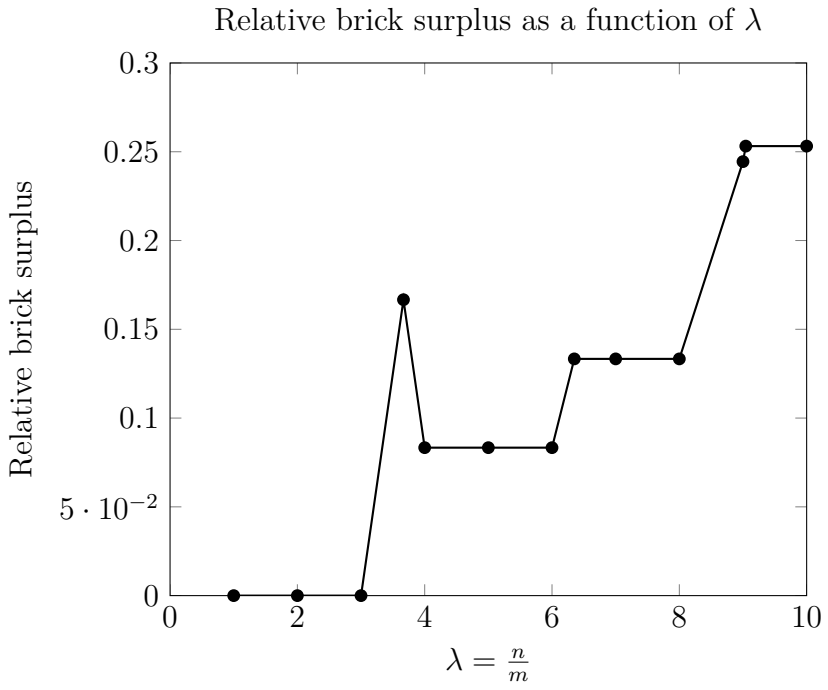

Figure B.1: Plot of relative brick surplus for small λ.

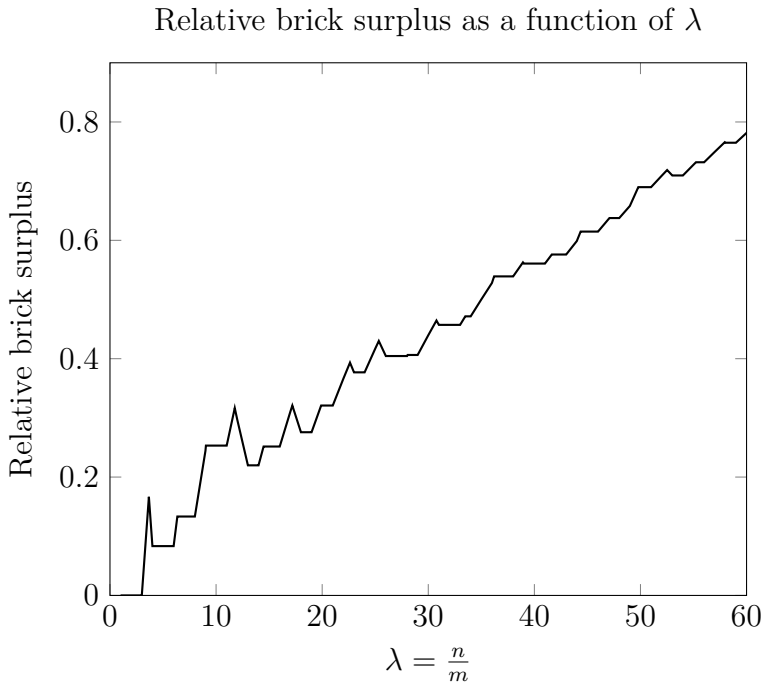

Figure B.2: Plot of relative brick surplus for large λ.

λ	surplus
1	0.000
2	0.000
3	0.000
4	0.083
5	0.083
6	0.083
7	0.133
8	0.133
9	0.244
10	0.253
11	0.253
12	0.297
13	0.220
14	0.220
15	0.252
16	0.252
17	0.310
18	0.276
19	0.276
20	0.321

λ	surplus
21	0.321
22	0.366
23	0.377
24	0.377
25	0.417
26	0.405
27	0.405
28	0.405
29	0.406
30	0.440
31	0.457
32	0.457
33	0.457
34	0.472
35	0.500
36	0.528
37	0.539
38	0.539
39	0.561
40	0.561

λ	surplus
41	0.561
42	0.576
43	0.576
44	0.599
45	0.615
46	0.615
47	0.636
48	0.638
49	0.658
50	0.690
51	0.690
52	0.709
53	0.710
54	0.710
55	0.728
56	0.732
57	0.749
58	0.765
59	0.765
60	0.782

Figure B.3: Table of approximate values of the relative brick surplus for integer values of λ.

bag cost	λ	surplus
1	3.667	0.167
2	6.350	0.133
3	9.044	0.253
4	11.761	0.317
5	14.477	0.252
6	17.188	0.321
7	19.902	0.321
8	22.622	0.393
9	25.338	0.430
10	28.052	0.406
11	30.772	0.465
12	33.490	0.472
13	36.206	0.539
14	38.923	0.563
15	41.642	0.576
16	44.360	0.615
17	47.076	0.638
18	49.795	0.690
19	52.514	0.719
20	55.231	0.732
21	57.948	0.766
22	60.668	0.793

Figure B.4: Table of approximate values of the relative brick surplus at the points when we stop using bags of given cost.

C. Values of f

Let us evaluate

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

for $\rho = 1.6$ and $k \in \{2, 3, \ldots, 60\}$. See Figure C.1 for both exact and approximate values of $f(k)$. You might notice that the signs of $f(k)$ are periodic (with the exception of first few values) with a period of 5.

k	$f(k)$	$\approx f(k)$	k	$f(k)$	$\approx f(k)$	k	$f(k)$	$\approx f(k)$
			21	$\mathbf{-\frac{11}{20}}$	**–0.55**	41	$\mathbf{-\frac{23}{40}}$	**–0.57**
2	2	2.00	22	$\frac{10}{21}$	0.48	42	$\frac{18}{41}$	0.44
3	0	0.00	23	$\mathbf{-\frac{6}{11}}$	**–0.55**	43	$\mathbf{-\frac{4}{7}}$	**–0.57**
4	1	1.00	24	$\frac{11}{23}$	0.48	44	$\frac{19}{43}$	0.44
5	$\frac{3}{4}$	0.75	25	$\frac{11}{24}$	0.46	45	$\frac{19}{44}$	0.43
6	$\mathbf{-\frac{2}{5}}$	**-0.40**	26	$\mathbf{-\frac{14}{25}}$	**–0.56**	46	$\mathbf{-\frac{26}{45}}$	**–0.58**
7	$\frac{2}{3}$	0.67	27	$\frac{6}{13}$	0.46	47	$\frac{10}{23}$	0.43
8	$\mathbf{-\frac{3}{7}}$	**-0.43**	28	$\mathbf{-\frac{5}{9}}$	**–0.56**	48	$\mathbf{-\frac{27}{47}}$	**–0.57**
9	$\frac{5}{8}$	0.62	29	$\frac{13}{28}$	0.46	49	$\frac{7}{16}$	0.44
10	$\frac{5}{9}$	0.56	30	$\frac{13}{29}$	0.45	50	$\frac{3}{7}$	0.43
11	$\mathbf{-\frac{1}{2}}$	**-0.50**	31	$\mathbf{-\frac{17}{30}}$	**–0.57**	51	$\mathbf{-\frac{29}{50}}$	**–0.58**
12	$\frac{6}{11}$	0.55	32	$\frac{14}{31}$	0.45	52	$\frac{22}{51}$	0.43
13	$\mathbf{-\frac{1}{2}}$	**-0.50**	33	$\mathbf{-\frac{9}{16}}$	**–0.56**	53	$\mathbf{-\frac{15}{26}}$	**–0.58**
14	$\frac{7}{13}$	0.54	34	$\frac{5}{11}$	0.45	54	$\frac{23}{53}$	0.43
15	$\frac{1}{2}$	0.50	35	$\frac{15}{34}$	0.44	55	$\frac{23}{54}$	0.43
16	$\mathbf{-\frac{8}{15}}$	**-0.53**	36	$\mathbf{-\frac{4}{7}}$	**–0.57**	56	$\mathbf{-\frac{32}{55}}$	**–0.58**
17	$\frac{1}{2}$	0.50	37	$\frac{4}{9}$	0.44	57	$\frac{3}{7}$	0.43
18	$\mathbf{-\frac{9}{17}}$	**-0.53**	38	$\mathbf{-\frac{21}{37}}$	**–0.57**	58	$\mathbf{-\frac{11}{19}}$	**–0.58**
19	$\frac{1}{2}$	0.50	39	$\frac{17}{38}$	0.45	59	$\frac{25}{58}$	0.43
20	$\frac{9}{19}$	0.47	40	$\frac{17}{39}$	0.44	60	$\frac{25}{59}$	0.42

Figure C.1: Table with exact and approximate values of $f(k)$ for $\rho = 1.6$ and $k \in \{2, 3, \ldots, 60\}$.