Dividing splittable goods evenly and with limited fragmentation

Algorithmica (2020) 82:1298–1328
https://doi.org/10.1007/s00453-019-00643-z

Dividing Splittable Goods Evenly and With Limited Fragmentation

Peter Damaschke[1]

Received: 7 May 2018 / Accepted: 15 October 2019 / Published online: 25 October 2019

Abstract
A splittable good provided in n pieces shall be divided as evenly as possible among m agents, where every agent can take shares from at most F pieces. We call F the fragmentation and mainly restrict attention to the cases $F = 1$ and $F = 2$. For $F = 1$, the max–min and min–max problems are solvable in linear time. The case $F = 2$ has neat formulations and structural characterizations in terms of weighted graphs. First we focus on perfectly balanced solutions. While the problem is strongly NP-hard in general, it can be solved in linear time if $m \geq n - 1$, and a solution always exists in this case, in contrast to $F = 1$. Moreover, the problem is fixed-parameter tractable in the parameter $2m - n$. (Note that this parameter measures the number of agents above the trivial threshold $m = n/2$.) The structural results suggest another related problem where unsplittable items shall be assigned to subsets so as to balance the average sizes (rather than the total sizes) in these subsets. We give an approximation-preserving reduction from our original splitting problem with fragmentation $F = 2$ to this averaging problem, and some approximation results in cases when m is close to either n or $n/2$.

Keywords Fair division · Weighted graph · Linear-time algorithm · Parameterized algorithm · Approximation algorithm

Mathematics Subject Classification 05C05 · 05C85 · 68P10 · 68Q25 · 68W25 · 68W40

This is a revised and largely extended version of a paper presented at the 42nd International Symposium on Mathematical Foundations of Computer Science MFCS 2017, LIPIcs vol. 83, 9:1–9:13. The approximation results are completely new.

✉ Peter Damaschke
ptr@chalmers.se

[1] Department of Computer Science and Engineering, Chalmers University, 41296 Göteborg, Sweden

1 Introduction

1.1 General Problem Statement

Suppose that n pieces of a good are given, with sizes $x_1, \ldots, x_n$. The good shall be divided among m agents, thereby respecting certain fairness criteria and restrictions. In a solution, let $y_1, \ldots, y_m$ denote the amounts that the agents receive, and let z_{ij} be the amount that agent j receives from piece i. Clearly, the conditions $y_j = \sum_{i=1}^{n} z_{ij}$ for all agents j and $\sum_{j=1}^{m} z_{ij} \leq x_i$ for all pieces i must be fulfilled. All mentioned variables are non-negative. The agents are identical.

Ideally we would like to divide the good evenly and completely, that is: $y_1 = \cdots = y_m$ and $\sum_{i=1}^{n} x_i = \sum_{j=1}^{m} y_j$. Without further restrictions it is a trivial problem to find mn numbers z_{ij} that satisfy these demands. But suppose that we also want to limit the fragmentation, in the following sense. Let F be some fixed positive integer. Every agent shall get parts of at most F distinct pieces. Formally, for every j we allow $z_{ij} > 0$ for at most F indices i. These indices can be chosen in the solution; only their number is limited by F. As opposed to this, every piece may be divided among an unlimited number of agents.

1.2 Motivations

We discuss a few motivations; for other natural applications see also [20] where a closely related problem is treated.

Imagine that some pieces of land, at n different locations and with areas $x_1, \ldots, x_n$, shall be assigned to farmers in a fair manner. Besides getting a fair share, it would be desirable for every single farmer to get only a few different fields, rather than several scattered ones, such that the farmer does not have to divide activities between many different locations. Only the amounts of land are of interest, but there are no specific geometric restrictions, as it is trivial to cut, e.g., a rectangle into smaller rectangles, once the desired sizes are decided.

One may also think of applications in scheduling, where the x_i are durations of n jobs that shall be divided among m workers, in such a way that they get equal workloads, and every worker is concerned with only a few different jobs, in order to limit context switching. Of course, in such a scenario we have to assume that the jobs can be arbitrarily split and also parallelized. To be more specific, an example where these assumptions are realistic is grading of the n exercises of an exam by m graders. Note that every grader dealing with an exercise has to become acquainted with that exercise, which causes extra work independent of the actual amount of solutions to grade. Therefore it seems appropriate to simply count the number F of different jobs (exercises), rather than adding fractions of graded solutions.

1.3 Related Topics

Prominent problems in Discrete Optimization, in various application domains, deal with cutting or connecting items from raw materials, e.g., the Cutting Stock and Skiving

Stock problems; see [17]. An action that is mathematically equivalent to cutting is packing items into containers, as in the Bin Packing and Knapsack problems [16]. (In fact, Skiving Stock is also known as Dual Bin Packing). However, the problems we consider here differ from all the mentioned problems in one way or another For instance, in the "Stock" problems we are given large amounts of items of only a few different sizes, and in the Bin Packing and Knapsack problems the sizes of items to be packed (or cut out) are prescribed.

Non-preemptive versions of scheduling with cardinality constraints have been considered, e.g., in [3,6,15].

There is a rich literature on fair division problems which became very popular under the name *fair cake-cutting*: A resource called a "cake" shall be divided among several agents, under various fairness requirements and geometric restrictions such as connectivity [2] or order. Some work assumes indivisible goods. The agents are not always identical but they may have different preferences and utilities as, e.g., in [5]. Much of the research in the field considers not only the division results but also the division process as a game, in settings where the intended divisions cannot be exactly realized, and different agents cut and choose pieces, e.g., in [7]. In envy-free divisions, agents have no incentives to change their own shares for others. We refer to a survey [19] of algorithms for cutting a cake represented by an interval.

The work being closest to ours is [20]; we discuss it below in Sect. 1.4 in the context of results.

1.4 Our Results

In Sect. 2 we formally define splitting problems where the good shall be divided completely and the minimum share shall be maximized (Max–Min Splitting) or vice versa (Min–Man Splitting). We call a solution perfect if all shares are equal.

In Sect. 3 we consider the case of fragmentation $F = 1$, which is solvable in linear time. One way[1] is to reduce both of our problems in linear time to the problem solved in [20]. Rephrased in our terminology, that problem requires to give equal shares of maximum size to all agents and waste the rest of the good. The optimum is found by linear-time selection in a set of candidate values. However, for our problems with full distribution of the good we independently developed an alternative approach (to generating the candidate set and proving correctness) which appears more intuitive and concise.

The main focus of the paper is, however, the case of fragmentation $F = 2$. First of all, notice that there is a "jump" from $F = 1$ to $F = 2$, in that the problem totally changes its quality: The restrictive demand $F = 1$ allows only very uneven solutions for some instances. In particular, if $m = n$, the only possible solution is to deliver one piece to every agent, regardless of their sizes, whereas $F = 2$ gives us the option to cut the large pieces. Interestingly enough, a perfect solution with $F = 2$ exists whenever $m \geq n - 1$, and it can even be constructed in a rather straightforward way. We expect that the differences in the problem's behavior between $F = 2$ and larger F are then less drastic.

[1] The author is indebted to the anonymous reviewer who pointed out this possibility.

Solutions for $F = 2$ can be naturally described by weighted graphs that we call solution graphs. As a side remark, this makes the case $F = 2$ a fractional version of the famous degree realization problem (see, e.g., [9] as an entry point, and [10] for a related editing problem) which requires to construct a graph whose vertex degrees equal a given sequence of integers. Based on the solution graphs, we present in Sect. 4 a linear-time elimination algorithm that finds, as said above, a perfect solution whenever $m \geq n - 1$, we prove strong NP-completeness for $m < n - 1$, and NP-completeness already for the special case when m is slightly smaller than $n - 1$. Moreover, the problem is NP-complete for any constant $F > 2$ as well.

It also follows that the problems cannot be fixed-parameter tractable in the parameter $n - m$, under standard hypotheses. However, this is different at the other end of the range of m: Note that $m \geq n/2$ is necessary for a solution to exist. Moreover, finding a perfect solution is trivial when $m = n/2$ (with n even). Thus it is natural to consider the problem parameterized by the "distance to triviality" $m - n/2$, or equivalently, $2m - n$. Indeed, we show that the problem of assigning equal amounts to all agents, or reporting that the instance does not allow that, is fixed-parameter tractable in the parameter $2m - n$.

In Sect. 5 we use the previous structural results to reduce Max–Min Splitting and Min–Man Splitting for $F = 2$ to a new problem that asks to divide a set of unsplittable items in bags (subsets) such that, roughly speaking, the average sizes in the bags are close to each other. The reduction preserves approximation ratios, but the reformulation with unsplittable items makes it easier to design approximation algorithms, as illustrated in Sect. 6. (A separate overview of these results is given in Sect. 6.) The problem resembles load balancing, but since averages rather than sums must be balanced, it also behaves quite differently.

2 Preliminaries

Throughout the paper, F is a fixed positive integer called the *fragmentation*. Common to all problem variants are the input and some of the constraints:
Splitting:
Given: n positive numbers $x_1, \ldots, x_n$, and an integer m.
Find: non-negative numbers z_{ij} and y_j (for $i = 1, \ldots, n$ and $j = 1, \ldots, m$) subject to:
$\forall j : \; y_j = \sum_{i=1}^{n} z_{ij}$,
$\forall i : \; \sum_{j=1}^{m} z_{ij} \leq x_i$,
$\forall j : \; |\{i \mid z_{ij} > 0\}| \leq F$,
and further constraints and objectives specified below.

The positive numbers z_{ij} and the numbers y_j are sometimes called *shares*. The problems require to find shares, or to recognize that no such numbers can satisfy the constraints. The above general constraints merely say that the share of every agent is the sum of shares from the different pieces, one can distribute at most the available amount of goods, and for every agent the fragmentation is bounded by F.

To fully specify the actual problems we only mention the additional constraints below, in order to avoid repetitions.

PERFECT SPLITTING: $\forall i : \sum_{j=1}^{m} z_{ij} = x_i$, and all shares y_j are equal.

We refer to a solution of PERFECT SPLITTING as a *perfect solution*. There, all agents get equal shares, and no goods are held back. The PERFECT SPLITTING problem is to find a perfect solution or to recognize that no perfect solution exists.

MIN–MAX SPLITTING: $\forall i : \sum_{j=1}^{m} z_{ij} = x_i$, and the largest share, $\max_j y_j$, is minimized.

MAX–MIN SPLITTING: $\forall i : \sum_{j=1}^{m} z_{ij} = x_i$, and the smallest share, $\min_j y_j$, is maximized.

In the last two problems, still all goods must be distributed, but in general the agents get different shares. If a perfect solution exists, then this is also the optimal solution to both MIN–MAX and MAX–MIN SPLITTING. The last two problems differ in their motivations: The MIN–MAX problem appears more appropriate for applications where some work must be divided completely, and the goal is not to load any individual agents too much. The MAX–MIN problem aims at giving everyone a guaranteed amount of a good, being as large as possible.

In the following we can always assume $m \geq n/F$, since otherwise there is obviously no solution that assigns all goods to agents. In particular, for $F = 1$ we assume that $m \geq n$ agents are present.

Another trivial remark is that we can, without loss of generality, multiply all sizes x_i by the same factor, e.g., in order to normalize their sum to some desired "convenient" value. The term *scaling* at several places in the paper refers to this action that does not change the given problem instance.

In the case $F = 2$ it is quite natural to represent any solution to a problem instance as a weighted graph:

Definition 1 The *solution graph* of a solution to (PERFECT, MIN–MAX, MAX–MIN) SPLITTING with $F = 2$ is a weighted graph with n vertices and m edges, specified as follows. We create a vertex of weight x_i for the ith piece. Every edge uv has two *ports* at the vertices u and v. The solution specifies a set of m edges and $2m$ weights of their ports. Specifically, if the jth edge has a port at the ith vertex, then the weight of this port is z_{ij}. The weight of the jth edge, y_j, is the sum of the weights of the two ports of the jth edge. Similarly, the weights of all ports at the ith vertex must sum up to x_i. An edge can also be a loop at one vertex, and in this case it is immaterial how its weight is divided into weights of the two ports.

Vertices and edges represent pieces and agents, respectively. Note that parallel edges may exist, since several agents may have shares from the same pieces. (Later we will see that parallel edges are not needed, but the definition allows them.)

The well-known SELECTION problem asks to find the tth smallest number in an unsorted set of n elements, for a prescribed number t. It can be solved in $O(n)$ time [4], with a relatively large hidden constant that can, however, be improved by randomization.

A problem with input size n and another input parameter k is fixed-parameter tractable (FPT) if some algorithm can solve it in $O(f(k) \cdot p(n))$ time, where f is some computable function of the parameter k, and p is some polynomial that must not depend on k. A problem *kernel* of an instance of an FPT problem is an equivalent

instance that can be computed in polynomial time and whose size depends only on the parameter. We refer to [18] for an introduction.

3 The Case Without Fragmentation (F = 1)

First we study MIN–MAX SPLITTING and MAX–MIN SPLITTING, which also subsumes the special case of PERFECT SPLITTING, for fragmentation $F = 1$.

Trivially, either of the first two problems always permits an optimal solution where, for every piece i, all $z_{ij} > 0$ are equal, since otherwise we can balance these values without making the solution worse. Thus, such a solution is fully characterized by a vector $(p_1, \ldots, p_n)$ with $\sum_{i=1}^{n} p_i = m$, where p_i is the number of indices j with $z_{ij} > 0$. For every such i, j we obviously have $y_j = z_{ij} = x_i / p_i$. This observation suggests the following definition.

Definition 2 We define Y to be the sequence of all numbers x_i / p, where $i = 1, \ldots, n$ and $p = 1, \ldots, m$, sorted in decreasing order. For $k = 1, \ldots, mn$ let $Y[k]$ denote the value at position k in Y.

The same value x_i / p may come from different i and p, therefore a value may occur multiple times in Y. However, we can break ties arbitrarily and sort equal values in any order. Formally this can also be achieved by random and infinitesimal perturbations of the values x_i. We will henceforth assume that all values in Y are distinct, hence Y is strictly monotone decreasing. This avoids circumstantial treatment of specific cases.

Intuitively, the p_i should be roughly proportional to the x_i. But the efficient computation of exact optimal solutions is less obvious and is addressed below. Note that we only count the time for solving the actual problem, i.e., for computing the numbers p_i, whereas the final "physical" division of goods among the m agents is not part of the computation. Adopting the unit cost measure where comparisons and algebraic operations with real numbers take constant time, the time bounds are independent of m which may be arbitrarily larger than n.

3.1 Maximizing the Minimum

The following Lemma 1 deals with the problem being inverse to MAX–MIN SPLITTING and yields a characterization of the optimal solution in Lemma 2.

Lemma 1 *For any fixed y with $0 < y \leq \min_i x_i$, let k be the maximum number of agents such that everyone can obtain an amount at least y. Then we have:*

(a) $k = \sum_{i=1}^{n} p_i$, *where* $p_i := \lfloor x_i / y \rfloor > 0$.
(b) k *is the maximum index with* $Y[k] \geq y$.
(c) $Y[k] = \min_i x_i / p_i$.

Proof (a) For every i we can divide the ith piece among at most $\lfloor x_i / y \rfloor$ agents. Summation over all i yields the assertion. Due to the assumption on y we have $p_i > 0$ for all i.

(b) Note that $x_i/s_i \geq y$ holds for all i and all $s_i \leq p_i$, and there exist k such pairs (i, s_i). Hence at least k members of Y are greater than or equal to y, that is, $Y[k] \geq y$. Since $x_i/(p_i + 1) < y$ holds for all i, we also see that no further members of Y have this property.

(c) As said above, the k values x_i/s_i ($s_i \leq p_i$) are exactly those members of Y being greater than or equal to y, that is, they are the first k items in Y, and clearly $\min_i x_i/p_i$ is the last of them. □

Lemma 2 *The optimal objective value for* MAX–MIN SPLITTING *with* $F = 1$ *equals*

$$\max \min_j y_j = \min\{Y[m], \min_i x_i\},$$

where the maximum is taken over all solutions.

Proof *Case 1* $Y[m] < \min_i x_i$. We apply Lemma 1 to $y := Y[m]$. Due to (b), the maximum number k of agents that can be served equals the maximum index k with $Y[k] \geq Y[m]$. This implies $k = m$. In other words, m agents can obtain an amount of at least $Y[m]$ each.

Assume that m agents can obtain more, say an amount of $y' > y$ each, where still $y' < \min_i x_i$. We apply Lemma 1 to y'. Due to (b), the maximum number k' of agents that can be served equals the maximum index k' with $Y[k'] \geq y' > y = Y[m]$. This implies $k' < m$. Hence we have shown by contradiction that m agents cannot obtain any amount larger than y.

Case 2 $Y[m] \geq \min_i x_i$. We apply Lemma 1 to $\min_i x_i$. Part (c) implies that $Y[k] = \min_i x_i/p_i \leq \min_i x_i \leq Y[m]$, thus $k \geq m$. That is, m agents can be served with an amount at least $\min_i x_i$. But $\min_i x_i$ is also a trivial upper bound on the objective value, which proves the assertion also in this case. □

It is trivial to check in $O(n)$ time whether all m agents can get shares of size $\min_i x_i$. If so, this solution is optimal, and we are done. From now on we can suppose that this trivial case has been checked negatively. Thus, we have $Y[m] < \min_i x_i$, and $Y[m]$ is the optimal value by Lemma 2.

For a good time bound we must find the value $Y[m]$ without naively generating all $m - 1$ preceding elements of Y. The intuition for a faster approach is to search for the optimal value near the average amount given to the agents.

Lemma 3 *Let* $\bar{y} := \sum_{i=1}^{n} x_i/m$ *be the average amount given to the m agents, and let k be the maximum number of agents such that every agent can actually obtain an amount at least* $\bar{y}$. *Then* $Y[k] \geq Y[m]$ *and* $m - k \leq n$.

Proof Clearly, $\bar{y} \geq \min_j y_j$ holds in any solution, hence $\bar{y} \geq \max \min_j y_j$ (where the maximum is taken over all solutions). Now, Lemma 2 implies $\bar{y} \geq Y[m]$, and Lemma 1 applied to $\bar{y}$ gives $Y[k] \geq \bar{y} \geq Y[m]$.

For $i = 1, \ldots, n$ we define $q_i := x_i/\bar{y}$, with integer part and fractional part $p_i := \lfloor q_i \rfloor$ and $r_i := q_i - \lfloor q_i \rfloor$, respectively. Lemma 1 (a) states that $k = \sum_{i=1}^{n} p_i$.

Thus we observe:

$$k = \sum_{i=1}^{n} p_i = \sum_{i=1}^{n} (q_i - r_i) \geq \sum_{i=1}^{n} q_i - n = m - n.$$

From this chain of inequalities we get $m - k \leq n$. □

Based on these inequalities we will now give an efficient algorithm for MAX–MIN SPLITTING. Remember that the trivial case was already excluded.

Algorithm MaxMin1

1. Compute $\bar{y} := \sum_{i=1}^{n} x_i/m$ and all $p_i := \lfloor x_i/\bar{y} \rfloor$.
2. Compute $k := \sum_{i=1}^{n} p_i$ and $Y[k] := \min_i x_i/p_i$.
3. If $m \leq k$ then $y := Y[k]$. Go to step 5.
4. If $m > k$ then:

4.1. Create the set R of the n ratios x_i/p_i sorted in decreasing order.
4.2. Mark the rightmost (i.e., smallest) element of R.
4.3. Move a pointer in R from left to right, i.e., starting at the largest ratio. For every ratio x_i/p_i encountered by the pointer, compute $p_i := p_i + 1$, compute x_i/p_i, and insert x_i/p_i at the correct position in the sorted set R.
4.4. Stop when the pointer is $m - k$ positions to the right of the marked element, and let y be the element found there.

5. Recompute all p_i by $p_i := \lfloor x_i/y \rfloor$.

Theorem 1 MAX–MIN SPLITTING *with* $F = 1$ *can be solved in* $O(n \log n)$ *time.*

Proof We run Algorithm MaxMin1. Steps 1 and 2 cost $O(n)$ time, and they are correct due to Lemma 1. By Lemma 2, the optimal value is $Y[m]$. We must show that the computed value y equals $Y[m]$. Lemma 3 ensures that $Y[k] \geq Y[m]$ and $m - k \leq n$. If $m \leq k$, then $Y[m] = Y[k]$, and Step 3 outputs the correct y. If $m > k$, then we only have to find the next $m - k$ members of Y after $Y[k]$, and since $m - k \leq n$, these are at most n further members.

We show that Step 4 generates $Y[k], \ldots, Y[m]$ and stops at $Y[m]$. Since $Y[k] = \min_i x_i/p_i$, the last element of the initial R in Step 4.2 (the marked element) is $Y[k]$. Whenever we insert a new ratio, its position in R is to the right of the pointer. For this reason, and because R comprises *all* elements of Y between the marked $Y[k]$ and the pointer, we are at $Y[m]$ when we stop.

To support fast insertions we host R in a priority queue. Thus Step 4 runs in $O(n \log n)$ time. Step 5 computes the final values p_i in $O(n)$ time. □

3.2 Minimizing the Maximum

In order to minimize $\max_j y_j$ we use the sequence Y from Definition 2 as well. For formal reasons we also set $Y[0] := \infty$. The scheme is pretty much the same as for MAX–MIN SPLITTING, but as the two problems are not symmetric, care must be taken for several details that are different.

Lemma 4 *For any fixed $y > 0$ that does not appear in Y, let $k \geq n$ be the minimum number of agents needed such that every agent has to take an amount at most y. This number k satisfies:*

(a) $k = \sum_{i=1}^{n} p_i$, *where* $p_i := \lceil x_i/y \rceil$.
(b) k *is the maximum index with* $Y[k-n] \geq y$.
(c) $Y[k-n] = \min_i x_i/(p_i - 1)$.

Proof (a) For every i we must split the ith piece among at least $\lceil x_i/y \rceil$ agents. Summation over all i yields the assertion.

(b) Note that $x_i/s_i \geq y$ holds for all i and all $s_i \leq p_i - 1$, and there exist $k-n$ such pairs (i, s_i). Hence at least $k-n$ members of Y are greater than or equal to y, that is, $Y[k-n] \geq y$. Since $x_i/p_i < y$ holds for all i, we also see that no further members of Y have this property.

(c) As seen above, the $k-n$ values x_i/s_i ($s_i \leq p_i - 1$) are exactly those members of Y being greater than or equal to y, that is, they are the first $k-n$ items in Y, and clearly $\min_i x_i/(p_i - 1)$ is the last of them. □

Lemma 5 *The optimal objective value for* Min–Max Splitting *with $F = 1$ equals*

$$\min \max_j y_j = Y[m-n+1],$$

where the minimum is taken over all solutions.

Proof We apply Lemma 4 to $y := Y[m-n+1]+\delta$ with an infinitesimal $\delta > 0$, added in order to meet the requirement that y itself does not occur in Y. Due to (b), The minimum number k of agents equals the maximum index k with $Y[k-n] > Y[m-n+1]$. This implies $k = m$. In other words, m agents have to take an amount of at most $Y[m-n+1]+\delta$ each. Since δ can be made arbitrarily small, their maximum load is bounded by $Y[m-n+1]$.

Assume that the m agents have to take even less, say $y' < Y[m-n+1]$. We apply Lemma 4 to y'. Due to (b), the minimum number k' of agents equals the maximum index k' with $Y[k'-n] \geq y'$. Since, in particular, $Y[m+1-n] \geq y'$, this implies $k' \geq m+1$. This shows that more than m agents are needed to bound their maximum load by any amount smaller than $Y[m-n+1]$. □

We have already seen that $Y[m]$ can be determined from $\bar{y}$ in $O(n \log n)$ time. Since n is known from the instance, an algorithm similar to MaxMin1 can also determine $Y[m-n+1]$, which yields the optimal solution for Min–Max Splitting due to Lemma 5. We can readily state:

Theorem 2 Min–Max Splitting *with $F = 1$ can be solved in $O(n \log n)$ time.*

3.3 Splitting in Linear Time

We have shown for both problems that the optimal value is found at a certain position in the sequence Y from Definition 2. For easier presentation we first gave $O(n \log n)$-time

algorithms. At a closer look, $O(n)$-time SELECTION can be used instead: Remember from Algorithm MaxMin1 and Theorem 1 that we only need to find the rth largest ratio (with $r := m - k > 0$) after $Y[k]$. A remark is that the $O(n \log n)$-time algorithms are likely to be faster for moderate n, due to the hidden constant factor in linear-time SELECTION.

Note that the sought element $Y[m]$ is not simply the rth largest ratio $x_i/(p_i + 1)$, since several ratios x_i/p with the same index i but varying p may be larger than $Y[m]$. Instead we will exploit a simple separation property:

Lemma 6 *For positive numbers x, x', p, q it is impossible that $x > x'$ and*

$$\frac{x}{q} > \frac{x'}{p} > \frac{x'}{p+1} > \frac{x}{q+1}.$$

Proof Clearly, the ratio of the two outer numbers is larger than the ratio of the two inner numbers:

$$\frac{x}{q} \cdot \frac{q+1}{x} > \frac{x'}{p} \cdot \frac{p+1}{x'}.$$

It follows $q < p$. But this implies $(q + 1)x' < (p + 1)x$, which contradicts the last inequality in the given chain. □

We need some precautions and special definitions to prepare for the algorithm. We find the maximum x_i in $O(n)$ time, and by renaming we assume that $x_1 = \max_i x_i$. For every positive integer q, the *bin* with *index* q is the interval $(x_1/q, x_1/(q + 1)]$. We define a simple procedure that takes as input a set S of ratios of the form x_i/p_i (p_i integer), generates new ratios, and puts them into the correct bins:

Procedure Gen(S)
For every ratio x_i/p_i in S, compute q with $x_1/q < x_i/(p_i + 1) \le x_1/(q + 1)$, and put $x_i/(p_i + 1)$ in the bin with index q.

Algorithm MaxMin1Lin
(Without loss of generality, assume that $x_1 = \max_i x_i$.)

1. Compute $\bar{y} := \sum_{i=1}^{n} x_i/m$ and all $p_i := \lfloor x_i/\bar{y} \rfloor$.
2. Compute $k := \sum_{i=1}^{n} p_i$ and $Y[k] := \min_i x_i/p_i$.
3. If $m \le k$ then $y := Y[k]$. Go to step 5.
4. If $m > k$ then:

4.1. Create the set S of the n ratios x_i/p_i. Run Gen(S).
4.2. Compute the index q of the bin containing $Y[k]$.
4.3. Repeat, until the processed bins together contain at least $m - k$ ratios smaller than $Y[k]$:
4.3.1. Create the set S of all ratios smaller than $Y[k]$, in the bin with index q.
4.3.2. Run Gen(S). Set $q := q + 1$.
4.4. Let s be the number of ratios smaller than $Y[k]$ in all bins except the current bin, $r := m - k$, and $t := r - s$.

4.5. Let y be the tth largest element in the current bin.
5. Recompute all p_i by $p_i := \lfloor x_i / y \rfloor$.

Theorem 3 MAX–MIN SPLITTING *and* MIN–MAX SPLITTING *with* $F = 1$ *can be solved in* $O(n)$ *time.*

Proof For MAX–MIN SPLITTING we run Algorithm MaxMin1Lin. Preparations and Steps 1–3 (which are identical to MaxMin1) take $O(n)$ time. Computing the index q of the bin that contains a given number requires $O(1)$ time, using divisions. Hence Gen(S) costs $O(|S|)$ time, which is $O(n)$ in Step 4.1.

We look for the rth largest ratio smaller than $Y[k]$. Due to Step 2, the ratios generated in Step 4.1 are smaller than $Y[k]$. Furthermore, Lemma 6 ensures that every bin contains at most one ratio x_i / p for every i. In particular, all ratios produced by Gen(S) from the set S in a bin are always put in later bins. From Lemma 3 we have $m - k \leq n$. Hence all runs of Gen in Step 4 take $O(n)$ time in total. Trivially, $O(n)$ time also suffices to count the ratios in Step 4.3 and to calculate s and t. The tth largest ratio in the final bin is our $Y[m]$, and Step 4.5 is done by a SELECTION algorithm in $O(n)$ time.

For MIN–MAX SPLITTING we can proceed similarly, since the only difference is the place of the optimal solution in Y (as specified in Lemma 5). □

4 Perfect Solutions for Fragmentation F = 2

In this section we study PERFECT SPLITTING when $F = 2$. Remember our notion of solution graphs from Definition 1 in Sect. 2.

First we characterize the instances of PERFECT SPLITTING that have a solution. An instance is given by n positive vertex weights $x_1, \ldots, x_n$ and an integer m. We scale the weights such that $\sum_{i=1}^{n} x_i = m$. Hence a perfect solution must satisfy $y_j = 1$ for all j. That is, all edge weights must be 1.

4.1 Many Agents Make it Easy

In the following result, a *forest*[2] is a graph without cycles, that is, a graph whose connected components are trees. Similarly to the time bounds for $F = 1$, we do not count the time for the trivial post-processing that actually splits the pieces. Therefore the time will not depend on m.

Algorithm Perfect2

1. Mark every vertex i *large* if $x_i > 1$, *normal* if $x_i = 1$, *medium* if $1/2 < x_i < 1$, and *small* if $0 < x_i \leq 1/2$.
2. While $m > n$ do:
   Attach a loop to some large vertex i, update $x_i := x_i - 1$ and $m := m - 1$.
3. If $m = n$ then:
3.1. If all vertices are normal, then append another loop to each, and stop.

[2] In the conference version a tree was claimed due to an inaccurate step, but there are counterexamples.

3.2. Else attach a loop to a large vertex i, update $x_i := x_i - 1$ and $m := m - 1$.
4. While $m = n - 1$ and $n > 2$ do:
4.1. Choose two vertices i and j with $x_i + x_j > 1$ and $x_j < 1$ as follows:
Either take two medium vertices i and j,
or take a large or normal vertex i, and a medium or small vertex j.
4.2. Join vertices i and j by an edge, where the port at j gets the weight x_j, and the port at i gets the weight $1 - x_j$. Update $x_j := 0$ and $x_i := x_i - 1 + x_j > 0$, hence $m := m - 1$ and $n := n - 1$.
5. If $m = 1$ and $n = 2$ then join the two vertices by an edge, where the ports get the weights of the vertices.

Theorem 4 *Every instance of* PERFECT SPLITTING *with* $F = 2$ *and* $m \geq n - 1$ *has a solution whose solution graph is a forest, possibly with loops attached to some vertices. Moreover, we can compute, for some solution, the edges of the forest and the number of loops at each vertex in* $O(n)$ *time.*

Proof We run Algorithm Perfect2 that constructs a solution graph. It successively creates certain edges of weight 1 and reduces the remaining weights x_i of the incident vertices accordingly. (Note that only the weights of the ports are needed to describe the solution, and the original x_i can be recovered afterwards). Similarly, m and n are used as variables to denote the number of edges yet to create, and the number of remaining vertices of positive weight, respectively.

As long as $m > n$, there exists a large vertex i, and we can do Step 2 until $m = n$ is reached. Clearly, we do not have to create the loops explicitly in $O(m)$ time, instead we compute the number ℓ_i of loops attached to every vertex i: Choose numbers $\ell_i \leq \lceil x_i \rceil - 1$ such that $\sum_i \ell_i = m - n$. This can be done by $O(n)$ arithmetic operations.

Next suppose that $m = n$. If all vertices are normal, then we are obviously done after Step 3.1. If not all vertices are normal, then some large vertex still exists, and Step 3.2 applies. Thus we reach $m = n - 1$ in this case.

From now on suppose that $m = n - 1$. For $n > 2$ we claim that the vertices i and j requested by Step 4.1 do exist. The correctness of Step 4.2 is then evident, and so is the correctness of Step 5.

In fact, since $m < n$, at least one medium or small vertex exists. If some large or normal vertex exists, too, then the claim is true. Thus suppose that all vertices are medium or small. If two medium vertices exists, the claim is also true. Thus suppose that all vertices are small, except at most one medium vertex. Now the equation $m = n - 1$ restricts the possibilities as follows.

If at least two small vertices exist, say the first two are small, then we have $x_1 + x_2 \leq 1$, and since also $x_i < 1$ holds for all $i \geq 3$, this is possible only if $n = 2$ and $x_1 = x_2 = 1/2$. If only one small vertex exists, then trivially $n = 2$ and $m = x_1 + x_2 = 1$. The case that no vertex is small is impossible, since then $n = 1$ and $m = x_1 > 0$. This way, each case leads to a contradiction, proving the claim.

In every step, the updates of weights take $O(1)$ time. The next edge is always built from two vertices from certain classes (large, normal, medium, small). Since arbitrary vertices from the respective classes can be chosen, it suffices to maintain four unsorted

sets of vertices. Thus we can also update these four sets and pick the needed vertices in $O(1)$ time. It follows an overall time bound $O(n)$.

As for the shape of the solution graph, consider the graph of edges inserted by the algorithm until any moment. We show the following invariant: Every connected component C of this graph retains at most one vertex with positive weight, and C is a tree, possibly with additional loops. This is trivially true in the beginning.

Consider any new edge that is not a loop. Prior to insertion of this edge, the weights of its vertices were positive, and upon insertion, the weight of exactly one of them drops to zero. Furthermore, since the two (positive) vertices were in two different connected components, the new edge merges these two connected components. Hence the invariant is preserved. It follows that the final solution graph is a forest, possibly with loops at some vertices. □

It is apparent from the proof Theorem 4 that an instance has, in general, many different solutions. One can argue that a small number of non-loop edges is preferable, as they correspond to agents that actually have to take shares from different pieces. Therefore we found it interesting, besides the time bound, that the solution graph can be forced to be a forest. It arises the question how difficult it is to compute a solution to PERFECT SPLITTING with $F = 2$ and $m \geq n - 1$ that also minimizes the number of non-loop edges. Note that we cannot simply attach loops as long as possible. For instance, if the vertex weights are 1.8, 0.1, 0.1, we can get two edges with weights $0.9 + 0.1$ at their ports, but if we begin with a loop, then the remaining instance 0.8, 0.1, 0.1 has no solution any more.

4.2 Structural Characterization and Hardness

Theorem 4 settles the case $m \geq n - 1$. In the following we also allow $m < n$ (but $m \geq n/2$, as said in Sect. 2). The conditions in Theorem 4 suggest the following concepts.

Definition 3 Let V be a set of elements, also called vertices, which are indexed by $1, \ldots, n$. (We identify elements with their indices.) Assume that every vertex i has a positive weight x_i. We call $I \subseteq V$ an *integral set* if $\sum_{i \in I} x_i$ is an integer. We call $I \subseteq V$ a *heavy set* if $\sum_{i \in I} x_i \geq |I| - 1$.

With this terminology, the instances allowing perfect solutions can be characterized as follows. Remember that the total amount of goods is m, and therefore every agent gets an amount of 1, without loss of generality.

Theorem 5 *An instance of* PERFECT SPLITTING *with $F = 2$ and a set V of n vertices of weights $x_1, \ldots, x_n$ where $\sum_{i=1}^{n} x_i = m$ (the number of agents) admits a solution if and only if V can be partitioned into heavy integral sets.*

Proof "only if": Suppose that there exists a perfect solution. Since $F = 2$, the solution can be represented as a solution graph G as in Definition 1, with vertex set V and with m edges (some of which may be loops). Let $C(k)$ denote the kth connected component of G; the indexing is arbitrary. Let n_k and m_k denote the number of vertices and edges,

respectively, in $C(k)$. Since every edge has weight 1, we get for each k that the vertex set V_k of $C(k)$ is integral. Specifically, the sum of vertex weights in $C(k)$ equals m_k. Due to connectivity we also have $m_k \geq n_k - 1$, thus V_k is a heavy set.

"if": Suppose that V has a partitioning into integral sets V_k which are also heavy. The latter means that $m_k \geq n_k - 1$, where n_k is the number of vertices of V_k, and m_k now denotes their total weight. We may consider every V_k as an instance of PERFECT SPLITTING with the given vertices and their weights, and with m_k agents. Due to $m_k \geq n_k - 1$ and Theorem 4, this instance has a solution with m_k agents. Since $m = \sum_k m_k$, the solutions to all these instances V_k combined form a solution to the entire instance with m agents. □

While PERFECT SPLITTING with $F = 2$ is easy for $m \geq n - 1$ due to Theorem 4, the complexity jumps when $m < n - 1$. The reason is that, unfortunately, it is hard to find a partitioning as required in Theorem 5, as we will show below. At first glance, hardness might appear counterintuitive, because with fragmentation $F = 2$ it should always be possible, within an elimination process as in Theorem 4, to take one piece and then the complementary amount (missing to sum 1) from some other piece. But the catch is that the remaining pieces might be too small to pair them up, and then the fragmentation $F = 2$ is not sufficient. Actually, we can show hardness by a reduction from 3- PARTITION. This problem is a natural candidate that has been used earlier for reductions to similar packing and scheduling problems as in [8].

In the following, note that a *multiset*, as opposed to a set, may contain several equal elements, and they are counted with their multiplicity, e.g., the multiset $\{x, x, y\}$ has three elements.

Theorem 6 PERFECT SPLITTING *with* $F = 2$ *is strongly NP-complete, and so are* MAX–MIN SPLITTING *and* MIN–MAX SPLITTING.

Proof We will give a polynomial-time reduction from the strongly NP-complete 3-PARTITION problem to PERFECT SPLITTING. We call any multiset with exactly three elements a *triplet*. An instance P of 3- PARTITION is a multiset with $3k$ positive rational numbers that shall be partitioned into k triplets such that the sum of the numbers in each triplet is the same. By scaling we can assume that the sum of the given numbers be k, hence the sum in each triplet must be 1. We can also assume $r \leq 1$ for all numbers r in P, since otherwise P has, trivially, no solution. We fix some small constant $d > 0$, in fact, any number d with $0 < d < 1/3$ will do.

For the reduction we take any given instance P with the above properties, and we transform every number r from P into $2(1/3 + dr)/(1 + d)$. Let Q be the multiset of these transformed numbers. They enjoy the following properties:

(a) Any three numbers from P sum up to 1 if and only if the three transformed numbers in Q sum up to 2.
(b) The sum of all numbers in Q is $2(3k/3 + dk)/(1 + d) = 2k$.

Let $n := 3k$ and $m := 2k$. Now we can view Q as an instance of PERFECT SPLITTING with $F = 2$, where n is the number of pieces, and $m = 2k$ is both the number of agents and the total amount to distribute.

Assume that P has a solution. Then each of its k triplets has the sum 1. Due to (a), the three transformed numbers in Q have the sum 2. Hence the triplets form a partitioning of Q into heavy integral sets. By Theorem 5, Q has a perfect solution.

Conversely, suppose that Q has a perfect solution. Using Theorem 5 again, Q can be partitioned into heavy integral sets. Since $n - m = k$ and the sets are heavy, the partitioning must consist of at least k sets.

Remember that $r \leq 1$ holds for all r from P. Hence any single number in Q is at most $2(1/3 + d)/(1 + d) < 1$. Any two numbers in Q have a sum at least $(4/3)/(1+d) > 1$ and at most $4(1/3+d)/(1+d) < 2$. Hence any integral set needs at least three elements. It follows that Q is partitioned into exactly k triplets. Using again that these sets are heavy, we see that the sum in each triplet is at least 2. Since, due to (b), the total sum equals $2k$, the sum in each triplet is exactly 2. Using (a) again, it follows that each corresponding triplet in P has the sum 1. That means, P has a solution.

Since PERFECT SPLITTING is a special case of the two optimization problems, the last assertion follows immediately. □

NP-hardness (but not necessarily in the strong sense) holds already when m is slightly smaller than $n - 1$, and the proof is less technical. Together with Theorem 4) this establishes a dichotomy.

Theorem 7 PERFECT SPLITTING *with* $F = 2$ *and* $m = n - t$ *is NP-complete for every constant* $t \geq 2$*, and so are* MAX–MIN SPLITTING *and* MIN–MAX SPLITTING. *Moreover, NP-completeness holds already for instances with sizes* $x_i < 1$ *for all* i.

Proof First let $t = 2$. Consider any instance where $m = n - 2$, and $x_i < 1$ for all i. Any partitioning into heavy integral sets necessarily consists of exactly two sets I and J, with $\sum_{i \in I} x_i = |I| - 1$ and $\sum_{j \in J} x_j = |J| - 1$. Due to Theorem 5, such an instance has a solution if and only if it can be partitioned into sets I and J with these properties.

On this basis we give a reduction from the NP-complete SUBSET SUM problem [8] to PERFECT SPLITTING. An instance of SUBSET SUM consists of positive rational numbers $y_1 \ldots, y_n$ and another value s, and the goal is to find a subset I of indices such that $\sum_{i \in I} y_i = s$. SUBSET SUM is NP-complete already in the case that $s = \sum_{k=1}^{n} y_k/2$. By scaling we can also assume that $\sum_{k=1}^{n} y_k = 2$, hence we have to divide the sum into $1 + 1$. Now we can also assume $y_k < 1$ for all k, since otherwise the instance has, trivially, no solution. Finally, we simply set $x_k := 1 - y_k$ for all k, to get an instance of PERFECT SPLITTING with $F = 2$ and $m = n - 2$. Equivalence of the instances of both problems is evident.

For every fixed $t > 2$ we construct the above instance and create $2(t - 2)$ further items $x_i = 1/2$. Since these additional items must form $t - 2$ pairs in any partitioning into heavy integral sets, equivalence is evident here, too. □

Theorem 8 PERFECT SPLITTING *is NP-complete for every constant fragmentation* $F > 2$*, and so are* MAX–MIN SPLITTING *and* MIN–MAX SPLITTING.

Proof This is established by a reduction from the case of fragmentation 2, with $x_i < 1$ for all i, which is NP-complete by Theorem 7. (We abandon the constant $t = n - m$.) In such instances, every agent must get shares from exactly two pieces.

For the moment we slightly tweak the problem and allow also dummy pieces of size 0. Given any instance with fragmentation 2 and sizes smaller than 1, we add $(F-2)m$ dummy pieces. The instance with fragmentation 2 has a perfect solution if and only if the resulting instance with fragmentation F has. The equivalence still holds if all dummy pieces have some equal size δ, where δ is some very small but positive number. □

4.3 Few Agents Make it Easy, Too

Inspecting the reductions that we used to show hardness results, one can notice that they produce instances with $m/n \geq 2/3$. Next we will see that PERFECT SPLITTING with $F = 2$ becomes "easy" if m is close to the smallest possible value $n/2$.

In the following we consider graphs and refer to any connected component with two vertices and one edge as a *pair*, and to any connected component with three vertices and two edges as a *triplet*. An *isolated vertex* is one that builds a connected component on its own. Note that the mentioned connected components have no loops. A neat small combinatorial observation on graphs will be used to bound the kernel size in the subsequent FPT result.

Lemma 7 *Let n and m be fixed, where $n/2 < m < (2/3)n$. Let $t = 2m - n$, and let q denote the number of vertices not being in pairs. Among all graphs that have exactly m edges and n vertices, none of them being isolated, there is a graph that maximizes q and only consists of pairs and t triplets. Consequently, $q \leq 3t$.*

Proof Consider any graph with m and n as specified. Since $m \leq n - 2$ (for $n \geq 6$), it has at least two connected components. Since $m < (2/3)n$, we even have that some connected component is a pair, because in all other connected components, the edges-to-vertices ratio is at least $2/3$.

Let C be some connected component that is not a tree. Then we can remove an edge from C such that C still remains connected. We re-insert this edge so as to merge two connected components. Since no vertices were isolated, this change does not create a new pair, hence q can only increase. Moreover, the number of connected components strictly decreases. The latter implies that, after a sequence of such changes, all connected components are trees.

If some connected component is a tree with at least four vertices, then we can remove a leaf and its incident edge. The remainder of the tree is still connected and is not a pair. We append the edge and the leaf to some pair (recall that a pair exists). Altogether this strictly increases q.

It follows that, in some graph with maximum q, all connected components are pairs and triplets. Let p and t' be the number of pairs and triplets, respectively. Clearly we have $m = p + 2t'$ and $n = 2p + 3t'$, hence $t = 2m - n = t'$. Since we have shown above that $q \leq 3t' = 3t$, we get that at most $3t$ vertices are not in pairs. □

Theorem 9 PERFECT SPLITTING *with $F = 2$ is fixed-parameter tractable (FPT) in the parameter $t = 2m - n$.*

Proof Consider an instance of PERFECT SPLITTING with $F = 2$ that has s solution. We also consider a partitioning as specified in Theorem 5 and refer to its heavy integral sets as *bags*. Let i and j be any two indices with $x_i + x_j = 1$.

Assume that i and j are in different bags, say, in $I \cup \{i\}$ and $J \cup \{j\}$, respectively, where $i \notin I$ and $j \notin J$. We replace them with two new bags $\{i, j\}$ and $I \cup J$. The pair $\{i, j\}$ is obviously a heavy integral set. Since the original bags were integral and the weight $x_i + x_j = 1$ has been removed, $I \cup J$ is also integral. The original bags were heavy, that is, $I \cup \{i\}$ and $J \cup \{j\}$ has weight at least $|I|$ and $|J|$, respectively. Since $x_i + x_j = 1$, we see that $I \cup J$ has a total weight at least $|I| + |J| - 1$, which means that this bag is heavy, too. Together this shows the existence of an alternative solution where $\{i, j\}$ is a bag.

Assume that i and j are in the same bag, but together with further indices, say, $K \cup \{i, j\}$ is a bag, where $K \neq \emptyset$ and $i, j \notin K$. We split it in two bags K and $\{i, j\}$. Clearly, K is integral. Since $K \cup \{i, j\}$ was heavy, it has a total weight at least $|K| + 1$. Hence, at least a weight of $|K|$ remains in K, which means that K is also heavy. This shows again the existence of an alternative solution where $\{i, j\}$ is a bag on its own.

We are ready to devise an FPT algorithm. First we pair up indices i and j with $x_i + x_j = 1$, as long as possible. Then we solve the remaining instance. That is, we search for a partitioning as in Theorem 5, of the set of indices that have not been paired up.

We claim that the given instance has a perfect solution if and only if this remaining instance has. The "if" direction is trivial. The "only if" statement holds due to the above exchange arguments: If a solution exists at all, then it can be transformed into a solution where any desired pair $\{i, j\}$ with $x_i + x_j = 1$ forms a bag. We also remark that, trivially, the pairs are uniquely determined up to isomorphism.

The pairing phase is easily implemented in $O(n \log n)$ time: Sort the list of weights, and then traverse it simultaneously in ascending and descending order.

If a solution graph exists then, by Lemma 7, the remaining instance can have at most $3t$ vertices. Hence, if more than $3t$ vertices are not paired up, we know that the instance has no perfect solution. In the positive case we may solve the remaining instance naively by exhaustive search. □

By way of contrast, Theorem 7 excludes an FPT (even an XP) algorithm in the parameter $b = n - m$, unless P=NP. In the next section we study approximation algorithms for fixed and small b or t.

5 Partitioning with Balanced Ratios

In this section we first introduce a seemingly very different problem, now with indivisible items. However, we show that it is closely related to SPLITTING, through the structural properties from Sect. 4. Besides, it may also be natural and interesting in its own right.

MIN–MAX (MAX–MIN) RATIO:

Given: n positive numbers $x_1 \geq \cdots \geq x_n$ (without loss of generality sorted in non-increasing order), and integers b and φ. We shall use the abbreviation $x := \sum_{i=1}^{n} x_i$.

Find: a partitioning of the index set $\{1, \ldots, n\}$ into b bags B_j $(j = 1, \ldots, b)$ so as to minimize (maximize) the maximum (minimum) of the ratios $s_j/(n_j - \varphi)$, where $n_j := |\{i : i \in B_j\}|$ denotes the number of indices in the jth bag, $s_j := \sum_{i \in B_j} x_i$ is the sum of the values x_i associated to these indices, and more than φ indices are in each bag, i.e., the constraint $\forall j : n_j > \varphi$ is satisfied. We refer to $s_j/(n_j - \varphi)$ as "the ratio of the bag" B_j.

The number φ is a constant "offset" in the denominator. The case $\varphi = 0$ requires to balance the averages in the bags, rather than the sums, as it is the case in load balancing problems. Imagine, for example, that the numbers x_i are scores indicating the strengths of individuals with respect to some skill, and a fixed number of temporary training teams shall be formed, where the teams should have similar average strengths (to obtain mixed teams where weaker members can learn from stronger ones, and the teams can compete), whereas the sizes of teams are less important. However, we were led to the problems by their connection to the respective SPLITTING problems: Positive numbers φ might appear artificial, but we will see below that the case $\varphi = 1$ provides a useful reformulation of the SPLITTING problems. We will consider generic but constant φ, as the offset does not change very much the structure and treatment of the problems.

Theorem 10 *For every instance of* MIN–MAX (MAX–MIN) SPLITTING *with* $F = 2$ *and* $b := n - m \geq 2$ *there exists an optimal solution where the set of pieces is partitioned into* b *bags such that: the* j*th bag has* $n_j > 1$ *pieces and is devoted to* $m_j = n_j - 1$ *agents,* $\sum_{j=1}^{b} n_j = n$*, and all agents assigned to the same bag get the same share.*

Proof Consider any solution and its solution graph. Clearly, it has at least b connected components. For every j, let the jth component contain n_j vertices and $m_j \geq n_j - 1$ edges (where $m_j \geq 1$ if $n_j = 1$). We refer to the vertex sets of the components as bags, and we note that m_j agents are assigned to the jth bag. Using Theorem 4, we can give all agents assigned to any one bag exactly the average amount in this bag. If some bag with $m_j \geq n_j$ exists, we can change the solution be merging this bag with another bag. Averaging over the merged bag can only improve the solution, that is, the maximum (minimum) share assigned to the agents can only decrease (increase). By doing such merge operations as long as possible, we end up with exactly b bags with the claimed properties. □

Theorem 11 *Any given approximation algorithm for* MIN–MAX (MAX–MIN) RATIO *with* $\varphi = 1$ *yields an approximation algorithm for* MIN–MAX (MAX–MIN) SPLITTING *with* $F = 2$*, with the same approximation ratio and asymptotic time bound.*

Proof Consider any instance $\mathcal{I}$ of MIN–MAX (MAX–MIN) SPLITTING with $F = 2$. We solve instead the instance $\mathcal{J}$ of MIN–MAX (MAX–MIN) RATIO with $\varphi = 1$, $b = n - m$, and with the given values x_i.

Theorem 10 yields that $\mathcal{I}$ has some optimal solution that translates into an optimal solution to $\mathcal{J}$ with the same objective value.

Now we take the approximate solution to $\mathcal{J}$ computed by the assumed approximation algorithm, say with approximation ratio c. Finally we transform it back into a

solution to $\mathcal{I}$ with the same objective value, by applying Algorithm Perfect2 separately to each bag B_j. This yields a c-approximate solution to $\mathcal{I}$.

Perfect2 runs in $O(n_j)$ time due to Theorem 4. Hence it needs $O(n)$ time for all bags, which affects the time bound for the approximation algorithm by at most a constant factor, since any algorithm trivially needs at least $\Omega(n)$ time to read the input. □

Theorem 11 provides a reduction that preserves the approximation ratio. Another view is that it separates the easy and hard part of the SPLITTING problems: While averaging within the bags is easy (Theorem 4), the difficulty is to find a partitioning into $b = n - m$ bags that balances the shares per agent across the different bags.

Due to Theorem 11 we can henceforth consider MIN–MAX (MAX–MIN) RATIO with $\varphi = 1$ in order to study approximation algorithms for MIN–MAX (MAX–MIN) SPLITTING. This turns out to be conceptually simpler (since we only have to take discrete decisions), and the simple translation using $b = n - m$ and Algorithm Perfect2 is given by Theorem 11.

6 Approximation Algorithms for MIN–MAX (MAX–MIN) RATIO

Since MIN–MAX SPLITTING and MAX–MIN SPLITTING with $F = 2$ and general m and n (where $n/2 \leq m \leq n - 2$) are strongly NP-complete due to Theorem 6, they do not allow FPTAS unless P=NP. However, we can still obtain various approximation results through MIN–MAX RATIO and MAX–MIN RATIO. We start with some preparations.

Let us scale any instance such that $\sum_{i=1}^{n} x_i = n$. We call a piece *large* if $x_i \geq 1$ and *small* if $x_i < 1$. For every large piece we define its *gain* to be $x_i - 1$. For every small piece we define its *loss* to be $1 - x_i$. These definitions naturally extend to subsets: Let g and ℓ be the sum of gains and losses, respectively, of the large and small pieces in a subset. We call $g - \ell$ the gain of the subset if $g \geq \ell$, and we call $\ell - g$ the loss of the subset if $\ell \geq g$. Note that both the gain and the loss of the set of all n pieces is zero.

For better readability, any fractional term which should actually be an integer (i.e., some number of elements) is rounded to the next integer, but we omit ceiling brackets. For any constant a we use that

$$1/(n-a) = (1/n)(n/(n-a)) = (1/n)(1 + a/(n-a)) = (1 + o(1))(1/n),$$
$$n/(n-a) = 1 + a/(n-a) = 1 + (1 + o(1))(a/n),$$

where $o(1)$ is some function that vanishes for n going to infinity.

The next lemma provides some simple bound.

Lemma 8 *Let φ and b be fixed integers. In every partitioning of a set of n pieces with $\sum_{i=1}^{n} x_i = n$ into b bags, there exists a bag with ratio at least (at most) $1 + (1 + o(1))(b\varphi/n)$. Consequently, this expression is a lower (upper) bound on the objective value of* MIN–MAX (MAX–MIN) RATIO.

Proof For the moment we re-scale the instance such that $\sum_{i=1}^{n} x_i = n - b\varphi$. Given a partitioning, we assign to every bag $n_j - \varphi$ "slots" (in case $\varphi = 1$ they correspond to

the edges in a solution graph) and imagine that the total size of all pieces in each bag is evenly divided among its slots. Hence the ratio of any bag equals the size per slot. Since we have $n - b\varphi$ slots in all bags together, clearly, there exist slots with sizes at least and at most 1, respectively. Finally we multiply the sizes again by the inverse scaling factor, namely $n/(n - b\varphi) = 1 + (1 + o(1))(b\varphi/n)$. □

These general bounds are also the best possible ones for both problems, witnessed by the simplest instance where $x_i = 1$ for all i. Provided that b divides n, the partitioning into b bags with n/b pieces gives all bags a ratio that matches our bound. Hence a general lower (upper) bound referring to only φ, b, n cannot be made larger (smaller) than that.

In the following we give approximation results for various types of instances. Here is an outline.

We mainly consider the situation when b/n is small, that is, when m is close to n. One may interpret b/n as the "distance to the easy case" of the SPLITTING problems (in Theorem 4). The concept of gain and loss allows us to formulate intuitive rules to balance the ratios of bags.

For MIN–MAX RATIO we must treat the cases when most pieces are small (Theorem 12) or large (Theorem 13) by different strategies of forming bags, but in both cases we arrive at approximation factors of the form $1 \pm \Theta(b/n)$ for any fixed φ (Corollary 1).

For MAX–MIN RATIO we get the same type of approximation results, but the problem has a quite different structure which requires different approaches and some other upper bounds besides that from Lemma 8. We begin with $\varphi = 0$ (Theorem 14 with Corollary 2), because this is also the basis of our result for any positive φ (Theorem 15). There remains a downside of the algorithm used: it demands at least $(\varphi + 1)b$ large pieces. By another modification we get rid of this extra assumption. We show this for $\varphi = 1$, the case being relevant for MAX–MIN SPLITTING (Theorem 16). Again, this solution makes use of the case $\varphi = 0$, mainly through an initial pairing of $2b$ pieces and some scaling.

Next we are wondering about approximation schemes that reduce the relative error $\Theta(b/n)$ to an arbitrarily small ε at the cost of higher running times. Using rounding and dynamic programming we obtain FPTAS for every fixed $b = n - m$, however with b in the exponent of the running time (Theorems 17 and 18).

Here, the rounding schemes need some attention, since we have to balance ratios in the end, and the numbers of pieces in the bags are not prescribed. The case of large b/n (up to $1/2$) requires different algorithms again, since the previous algorithms are tailored to small b/n. and their approximation guarantees would be poor or even meaningless. By other rounding schemes we obtain FPTAS also for $\varphi = 1$ and every fixed $t = 2m - n$, however with t in the exponent of the running time (Theorems 19 and 20). Once more, some detail of the analysis hinges on a bound shown earlier for the case $\varphi = 0$.

6.1 MIN–MAX RATIO, Mostly Small Pieces

Theorem 12 *Every instance of* MIN–MAX RATIO *with* $\sum_{i=1}^{n} x_i = n$*, where the majority of pieces are small, permits a solution with objective value at most* $1+(1+o(1))b/n$ *if* $\varphi = 0$*, and at most* $1 + (1 + o(1))(4\varphi + 2)b/n$ *if* $\varphi > 0$*, and such a solution can be constructed in* $O(n)$ *time.*

Proof First we compute the median and separate the $n/2$ largest from the $n/2$ smallest pieces. We first put some small pieces, below the median, into one bag, and we stop as soon as the loss of this bag exceeds φ.

Let p and s, respectively, be the number of pieces and the sum of their values x_i in the formed bag. Since the loss of every piece is smaller than 1, the loss $p - s$ of the bag satisfies $\varphi < p - s < \varphi + 1$. We also see that the ratio $s/(p - \varphi)$ of the bag is smaller than 1.

We iterate this process, that is, we put further small pieces below the median in another bag until its loss exceeds φ. We repeatedly build more bags until one of these cases appears:

(1) The number of such bags reaches $b' := b - 1$. Then we put all remaining pieces in the bth bag.
(2) Half of the pieces are used up. Then we abort the process, leaving the current bag unfinished at the moment.

All bags formed so far consist exclusively of small pieces. If case (1) applies, we have created b' bags, with at most $n/2$ pieces and a loss of at most $b'(\varphi + 1)$ in total. The remaining $q \geq n/2$ pieces are put in the bth bag. Trivially, its gain is at most $b'(\varphi+1)$. This bounds the sum of the sizes of pieces in this bag by $q + b'(\varphi + 1)$. The ratios of the other bags are bounded by 1, as seen above. The ratio of the last bag is now at most

$$\begin{aligned}(q + b'(\varphi+1))/(q-\varphi) &= 1+(b'(\varphi+1)+\varphi)/(q - \varphi) \leq 1+ v(b\varphi + b')/(n/2 - \varphi)\\ &= 1 + 2(b\varphi + b')/(n - 2\varphi).\end{aligned}$$

If case (2) applies, the $n/2$ smallest pieces also have together a loss at most $b'(\varphi+1)$, hence the total gain of the $n/2$ largest pieces is at most $b'(\varphi + 1)$, too. The smallest of the latter pieces (the median) has therefore a size at most $1 + 2b'(\varphi + 1)/n$. It follows trivially that the size of each of the $n/2$ smallest pieces is bounded by this term.

Now we change the partitioning in case (2) as follows. We create one bag of the $n/2$ largest pieces and divide the $n/2$ others arbitrarily into b' bags, each with $n/(2b')$ pieces. The ratio in the large bag is at most $1 + 2(b\varphi + b')/(n - 2\varphi)$ by the same calculation as before (with $q = n/2$). The ratio in each of the small bags is at most

$$\begin{aligned}&(1 + 2b'(\varphi + 1)/n)(n/(2b'))/(n/(2b') - \varphi) = (n + 2b'(\varphi + 1))/(n - 2b'\varphi)\\ &= (n - 2b'\varphi + 4b'\varphi + 2b'))/(n - 2b'\varphi) = 1 + (4\varphi + 2)b'/(n - 2b'\varphi).\end{aligned}$$

The proved upper bounds on the ratios of bags are:

$$1, 1 + 2(1 + o(1))(b\varphi + b')/n, 1 + (1 + o(1))(4\varphi + 2)b'/n.$$

The latter expression is asymptotically the maximum. With $b' < b$ we finally get the simpler term $1 + (1 + o(1))(4\varphi + 2)b/n$.

For $\varphi = 0$ we get a better result with a much simpler construction and calculation: Each of the b' smallest pieces forms a bag on its own, and the $n - b'$ largest pieces are put in one bag. Since the gain in the large bag is smaller than b', its ratio is bounded by $n/(n - b') = 1 + (1 + o(1))b/n$.

The time is obvious; note that no sorting is required. □

6.2 Min–Max Ratio, Mostly Large Pieces

For the sake of the following lemma, temporarily some x_i are allowed to be negative.

Lemma 9 *A set of n pieces of sizes $x_1 \geq \cdots \geq x_n$, with $x_1 > 0$, can be partitioned into b bags such that every bag contains exactly $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$ pieces, and the sums differ pairwise by at most d, where $d = x_1$ if $x_n \geq 0$, and $d = x_1 - x_n$ if $x_n < 0$. Such a partitioning can be constructed in $O(n \log b)$ time.*

Proof We assume that b is a divisor of n, otherwise we additionally create $b\lceil n/b \rceil - n$ dummy pieces with $x_i = 0$. We partition the set of pieces arbitrarily into n/b subsets of b pieces. Starting from b empty bags we apply the following procedure n/b times: sort the bags in descending order of sums, sort the next b pieces in ascending order, and add one piece to each bag, in this order. It is straightforward to verify the invariant that, after every such step, the sums in the bags differ pairwise by at most d. The time is dominated by sorting the bags n/b times in $O(b \log b)$ time. □

Theorem 13 *Every instance of* Min–Max Ratio *with $\sum_{i=1}^{n} x_i = n$, where the majority of pieces are large, permits a solution with objective value at most $1 + (1 + o(1))(2\varphi + 4)b/n$, which can be constructed in $O(n \log b)$ time.*

Proof Initially we let every piece be a bag on its own. Then we successively add single small pieces arbitrarily to bags that have a gain, as long as each of them retains some gain. That is, we stop only when no further small pieces can be added to any bag without creating a loss. The emptied bags are deleted.

Since the majority of pieces were large, after this procedure we still have at least $n/2$ bags. No bag has a gain larger than 1, because otherwise we can do another step: Since the sum of gains equals the sum of losses, there still remains some single piece with a loss, and since its loss is smaller than 1, we can add it to the alleged bag.

Now we treat the current bags as "super-pieces" and use Lemma 9 to put them into exactly b bags. However, instead of the sizes but we use the gains in the role of the values x_i, where a loss ℓ counts as a negative gain $-\ell$.

Eventually, every bag contains at least $n/(2b)$ super-pieces, thus at least $n/(2b)$ pieces. The gains of the super-pieces differ pairwise by at most 2, hence so do the gains of the final bags. Since the sum of gains is zero, it follows that the gain of every bag is at most 2. Altogether this bounds the ratios by $(n/(2b) + 2)/(n/(2b) - \varphi) = (n + 4b)/(n - 2b\varphi)$, which can be written as $(n - 2b\varphi + 2b\varphi + 4b)/(n - 2b\varphi) = 1 + (1 + o(1))(2\varphi + 4)b/n$.

The time bound comes from Lemma 9. □

6.3 Min–Max Ratio, Approximation Results

Corollary 1 Min–Max Ratio *can be solved within the following approximation factors:*

- $1 + (1 + o(1))b/n$ *in* $O(n)$ *time, if* $\varphi = 0$ *and most pieces are small,*
- $1 + (1 + o(1))(3\varphi + 2)b/n$ *in* $O(n)$ *time, if* $\varphi > 0$ *and most pieces are small,*
- $1 + (1 + o(1))(\varphi + 4)b/n$ *in* $O(n \log b)$ *time, if most pieces are large.*

Proof Divide the upper bounds from Theorems 12 and 13 by the general lower bound from Lemma 8. □

The true approximation ratio (compared to the optimal solution of the respective instance) might be even closer to 1. One indicator that there might exist fast algorithms improving upon the results of Corollary 1 is that the algorithm in Theorem 12 appears a little counterintuitive. It may be more natural to mix large and small pieces in a bag. (On the other hand, for the maximum ratio it does not matter too much whether the largest pieces are in the same bag or in different bags.) In any case, an improved result would also require stronger lower bounds being sensitive to the instances.

6.4 Max–Min Ratio

We remind the reader that the sizes of pieces are sorted: $x_1 \geq \cdots \geq x_n$.

Proposition 1 *In every partitioning into* b *or more bags, the* b*th largest average of the bags is at most* x_b*. In particular,* x_b *is an upper bound on the objective value of* Max–Min Ratio *with* $\varphi = 0$*.*

Proof We call the b largest pieces major, and the others minor. Consider the b bags with the b largest averages, where ties are broken arbitrarily. If some of these b bags contains only minor pieces, then obviously the average in this bag is at most x_b. The other case is that each of these b bags contains exactly one major piece. Then the average in the bag with a major piece of size x_b is again at most x_b. The second assertion follows trivially. □

Theorem 14 *Every instance of* Max–Min Ratio *with* $\varphi = 0$ *and with* $\sum_{i=1}^{n} x_i = n$ *permits a solution whose objective value is at least* $\min\{x_b, 1 - (1 + o())(4b - 2)/n\}$*. It can be constructed in* $O(n)$ *time.*

Proof Let $a \geq 1$ be the largest index with $a \leq b$ and $x_a \geq 1$; note that a exists. Initially we put the b largest pieces in the b bags, i.e., one in each bag. Then we successively add single pieces arbitrarily to the first a bags, as long as each of them retains some gain. That is, we stop only when no more pieces can be added to any bag without creating a loss.

Assume that still a or more pieces are outside the bags. Since the total loss of these pieces is at most the total gain of the first a bags, we get that the smallest loss of such a piece is at most the largest gain among the first a bags. But then we can add another piece to some bag, contradicting the stop criterion.

Hence some number $q < a$ of pieces remain outside the bags. In the final phase we will assign them to the first a bags in a special way. Let p denote the total number of pieces in the first a bags. Since more than $n-a$ pieces are now in the bags, and each of the $b-a$ last bags holds exactly one piece, we have $p > n-a-(b-a) = n-b$.

Let p_j denote the number of pieces in the jth bag, $j \le a$. Remember that all these bags still have a gain, and the loss of every piece is smaller than 1. Hence, if we assign q_j further pieces to the jth bag, then its loss is at most q_j, and therefore its ratio is still at least $1 - q_j/(p_j+q_j) \ge 1 - q_j/p_j$. This lower bound is also valid if $q_j = 0$. Now observe that maximizing $\min_j(1-q_j/p_j)$ is equivalent to maximizing $\min_{j|q_j>0} p_j/q_j$, under the constraint $\sum_{j=1}^{a} q_j = q$.

Specifically we proceed as follows. Assume $p_1 \ge \cdots \ge p_a$ by re-indexing. Initially let $q_1 := 1$ and $q_j := 0$ for all $j > 1$. Let k be the smallest index such that $q_k = 0$ (hence initially $k = 2$). We successively add single pieces to the bags according to the following rule. We pick an index $j < k$ with largest p_j/q_j. If $p_j/(q_j+1) \ge p_k$ then we set $q_j := q_j + 1$, else we set $q_k := 1$ followed by $k := k+1$. This step is repeated until all pieces are assigned.

We have the following invariant for all $j < k$: all p_j/q_j differ pairwise by factors at most 2, and $p_j/q_j \ge p_k$. This is vacuously true in the beginning, and every step preserves the invariant (in either branch of the if-then-else clause) since $p_j/(q_j+1) \ge p_j/(2q_j)$ and $p_k \ge p_{k+1}$.

In the final state, p_k is the largest number of pieces of a bag that did not receive further pieces. From our invariant we can conclude several things. The q considered pieces have been assigned to bags containing a total of at least qp_k pieces. The other $a-k+1$ bags that have not received further pieces (and thus still have a gain) contain a total of at most $(a-k+1)p_k$ pieces. Hence the former bags (that now have a loss) contain at least a fraction $q/(q+a-k+1)$ of the $p > n-b$ original pieces in our a bags. If all p_j/q_j were equal, their value would be at least $(pq/(q+a-k+1))/q = p/(q+a-k+1) > (n-b)/(2a-1) \ge (n-b)/(2b-1)$, where we used $q < a \le b$. Since the various p_j/q_j differ by factors at most 2, the smallest one is still larger than $(n-b)/(4b-2)$. That is, $\min_{j\le a}(1-q_j/p_j) > 1-(4b-2)/(n-b)$, which yields the claimed result. In the case $a < b$ (equivalently $x_b < 1$) we notice that the other $b-a$ bags still contain only single pieces whose smallest size equals x_b.

As for the time bound $O(n)$; note that we need not sort the pieces. □

Corollary 2 MAX–MIN RATIO *with* $\varphi = 0$ *and* $\sum_{i=1}^{n} x_i = n$ *can be solved to optimality if* $x_b < 1 - (1+o(1))(4b-2)/n$, *and otherwise within a factor* $1-(1+o(1))(4b-2)/n$ *of optimum, in* $O(n)$ *time.*

Proof This follows instantly from Theorem 14, the upper bound x_b from Lemma 1, and the trivial upper bound 1. □

Theorem 15 MAX–MIN RATIO *with* $\varphi > 0$, $\sum_{i=1}^{n} x_i = n$, *and* $x_{(\varphi+1)b} \ge 1$ *can be solved within a factor* $1-(1+o(1))(4b-2+(b-1)\varphi)/n$ *of optimum in* $O(n)$ *time.*

Proof We proceed as in Theorem 14 and only discuss the modifications. Due to the assumption $x_{(\varphi+1)b} \ge 1$ it is trivial to build b initial bags, each with exactly $\varphi+1$ pieces and a gain. Then we successively add the other pieces to the bags, following literally the

procedure in the proof of Theorem 14 and temporarily disregarding the positive offset φ. Precisely as before, $1-(4b-2)/(n-b)$ is a lower bound on the minimum ratio. Since, trivially, a bag cannot have more than n pieces, the offset φ raises this bound by a factor at least $n/(n-\varphi)$. This yields a lower bound $1-(1+o(1))(4b-2-\varphi)/n$ on the ratios. Together with the upper bound $1+(1+o(1))b\varphi/n$ from Lemma 8 we obtain the claimed result. □

6.5 Max–Min Ratio for $\varphi = 1$, Unrestricted

The assumption $x_{(\varphi+1)b} \geq 1$ in Theorem 15 can be weakened, but then it becomes more technical. Actually we only need to be able to quickly construct a partitioning of the largest $(\varphi+1)b$ pieces into b bags with exactly $\varphi+1$ pieces and a gain. We give a modified algorithm specifically for $\varphi = 1$, but now for arbitrary values x_i. The following quantity plays a central role.

Definition 4 We define $\sigma := \min_{1\leq j\leq b} x_j + x_{2b+1-j}$.

Next we prove σ to be an upper bound in the case of $\varphi = 1$.

Lemma 10 *In every partitioning into b or more bags with at least two pieces each, the bth largest average of the bags is at most $\sigma/2$. In particular, σ is an upper bound on the objective value of* Max–Min Ratio *with $\varphi = 1$.*

Proof The second assertion follows indeed from the first one, since with $\varphi = 1$, the ratio of a bag is at most twice its average. Now we prove the first assertion.

We call the $2b$ largest pieces *major*, and the others *minor*. The *2-average* of a bag is the average of sizes of its two largest pieces. The *principal* bags are the b bags with largest 2-averages. (Ties are broken arbitrarily.)

We claim that the bth largest 2-average is at most $\sigma/2$. Then, so is the bth largest average, by the following argument: The average of a bag is at most its 2-average. Hence there cannot exist b averages exceeding the bth largest 2-average.

To prove the claim we consider the majors in the b principal bags. If some principal bag has no majors, then its 2-average is, trivially, at most $\sigma/2$, and we are done. If some principal bag has more than two majors, then we move its smallest major to some principal bag with only one major (which must exist), thus raising the 2-averages of both bags. After exhaustive application of this rule, every principal bag has either one major or two majors. Assume that a major exists in some bag B' that is not principal. Then some principal bag B has only one major. We exchange a minor of B and a major of B'. This can only increase the 2-average of B and decrease the 2-average of B'. After exhaustive application of this rule, we have got exactly two majors in every principal bag. We have never changed the set of principal bags nor decreased any 2-average therein. Finally, another straightforward exchange argument shows that the smallest 2-average in the principal bags is maximized if each x_j is paired up with x_{2b+1-j}. □

Theorem 16 Max–Min Ratio *with $\varphi = 1$ and $\sum_{i=1}^{n} x_i = n$ can be solved to optimality if $\sigma < 1-(1+o(1))(3b-2)/n$, and otherwise within a factor $1-(1+o(1))(4b-2)/n$ of optimum, in $O(n)$ time.*

Proof We slightly modify the algorithm from Theorem 14: Initially we pair up the $2b$ largest pieces to form b bags, each of them having two pieces, and with minimum sum σ. We also pretend that these two pieces in each bag form only one piece whose size is the sum of the two. Thus we can henceforth work with $\varphi = 0$ instead of the true $\varphi = 1$. Since the number of pieces is now $n - b$, we multiply all sizes with the scaling factor $(n - b)/n = 1 - b/n$, such that the sum of all sizes equals $n - b$.

We sort the b bags by descending sums and continue exactly as in the proof of Theorem 14. Note that now $(1 - b/n)\sigma$ gets the role x_b had in Theorem 14. Thus we obtain a solution whose objective value is at least $\min\{(1-b/n)\sigma, 1-(1+o(1))(4b-2)/(n-2b)\}$.

To recover the original sizes we eventually re-scale by the factor $n/(n - b) = 1 + b/(n - b)$ and obtain $\min\{\sigma, 1 - (1 + o(1))(3b - 2)/n\}$. Finally we invoke the upper bounds σ and $1 + (1 + o(1))b/n$, from Lemmas 8 and 10. □

6.6 Approximation Schemes for Few Bags

Min–Max Ratio vaguely resembles the minimum makespan scheduling problem which is well studied. It is also strongly NP-complete, but it has a PTAS, and even an FPTAS when the number of machines (here corresponding to the number b of bags) is fixed; see [11–13]. However, the objectives are also quite different: The goal in makespan minimization is to balance the b sums, whereas we wish to balance, roughly, the averages in the bags. The knapsack problem admits a well-known FPTAS, too, running in $O(n^3/\varepsilon)$ time [14].

Those approximation schemes follow some basic ideas: Every input value (here x_i) is rounded to the next smaller or larger value in some limited set of discrete values. The rounded instance can then be solved to optimality in reasonable time by dynamic programming. The actual solution is recovered by memoization and backtracing in the standard way. Replacing the rounded values in the solution by their original values finally causes some approximation error. The finer the discrete set is, the larger is the time but the smaller is the error. The challenge is now to adapt the details of this technique to Min–Max Ratio and Max–Min Ratio.

Two natural rounding regimes come to mind: The discrete set could be the set of integer multiples of some fixed $\delta > 0$, or a sorted set whose consecutive members differ by factors at most $1 + \varepsilon$, for some fixed $\varepsilon > 0$ (e.g., powers of $1 + \varepsilon$). If the rounded x_i are multiples of δ, then so are the sums of all possible subsets. Moreover, the ratios of all possible bags have absolute errors at most δ. In the latter regime, the rounded x_i have relative errors at most ε, and hence, so have the ratios of all possible bags.

The dynamic programming phase would construct bags by successively adding pieces. We only need to keep track of the numbers of pieces and the sums of their sizes in the bags.

In the following we will first use an equidistant discrete set.

Lemma 11 *For any given instance of* Min–Max (Max–Min) Ratio *with* $\sum_{i=1}^{n} x_i = n$*, and for any given* $\delta > 0$*, a solution that deviates from the optimum by an absolute error of at most* δ *can be obtained in* $O((n^2/\delta)^b)$ *time.*

Proof For the given $\delta > 0$ and for every index i, let v_i be the unique integer with $v_i\delta \leq x_i < (v_i + 1)\delta$. We round every x_i, that is, we replace it with $v_i\delta$. We also arrange the pieces in an arbitrary order.

Then we do standard dynamic programming. We assign the pieces (in the fixed order) to the b bags in all possible ways. However, we do not store the actual assignments but only a vector of $2b$ numbers: the number of pieces and the sum of their rounded values, in each of the b bags, and if the same vector is produced multiple times, we retain only one copy.

Finally we take the optimal rounded solution according to the objective function at hand (minimizing the maximum or maximizing the minimum ratio, for the given φ). Invalid solutions having fewer than $\varphi + 1$ pieces in some bag are discarded.

Since the largest possible sum in every bag is trivially n, we need only $O(n/\delta)$ discrete values. Thus there exist only $O(n^b(n/\delta)^b/b!)$ different vectors. As already said, the absolute errors of the averages of the bags are at most δ. If $\varphi > 0$, the absolute errors of the ratios of the bags can be larger by some constant factor, but then we work with some smaller step length $\Theta(\delta)$ instead, which adds only a constant factor to the time bound. The time for dynamic programming is b times the number of vectors. □

Theorem 17 Min–Max Ratio *can be solved within a factor* $1 + \varepsilon$ *of optimum in* $O((n^2/\varepsilon)^b)$ *time.*

Proof We apply Lemma 11 with $\delta := \varepsilon$ and observe that the absolute error of the objective also bounds its relative error, since the objective value is trivially at least 1. □

For Max–Min Ratio the way is the same, with only slightly different details: Now, the previous results (that culminated in Corollary 2 and Theorem 16) ensure that either the problem can be solved to optimality, or the objective is at least some large fraction of 1, such that it suffices again to apply Lemma 11, with some $\delta := \Theta(\varepsilon)$. This yields:

Theorem 18 Max–Min Ratio *with* $\varphi \leq 1$ *can be solved within a factor* $1 - \varepsilon$ *of optimum in* $O((n^2/\varepsilon)^b)$ *time.*

6.7 Approximation Schemes for Many Small Bags

We finally consider the case of $\varphi = 1$ and large b/n, close to the threshold $1/2$. It is more convenient to use the parameter $t := 2m - n$ again. We will get a FPTAS for any fixed t. As one ingredient we need a hybrid of the two standard rounding regimes discussed in the previous section.

Lemma 12 *Let* $\delta > 0$ *be fixed. Consider a set of positive real numbers with an average of at least* 1. *Let us round every number smaller than* 1 *to the next integer multiple of* δ, *and every number larger than* 1 *to the next power of* $1 + \delta$ *with an integer exponent. (We may round upwards or downwards arbitrarily.) Then the average of the rounded values has a relative error* $O(\delta)$, *compared to the true average.*

Proof Say we have k numbers in our set. The absolute error of the sum of all numbers below 1 is $O(k\delta)$. Since the overall sum is at least k, the numbers below 1 contribute a relative error $O(\delta)$ to the overall sum. So do the numbers above 1, due to their rounding. Hence the relative error of the sum (and of the average) is still $O(\delta)$. □

Rephrasing Lemma 7, at most $3t$ pieces are in bags with more than two pieces, and at most t such bags can exist. In analogy to our FPT result for PERFECT SPLITTING (Theorem 9) we refer to the set of these bags as the "kernel" (somewhat abusing this term).

Theorem 19 MIN–MAX RATIO *with* $\varphi = 1$ *can be solved within a factor* $1 + \varepsilon$ *of optimum in* $(\Theta(1) \log t \cdot \varepsilon^{-1})^{4t} \cdot O(n)$ *time.*

Proof We scale the given instance of MIN–MAX RATIO such that $x_1 = t$. Since, in particular, the bag with the largest piece has a number of pieces limited by $O(t)$, its average and ratio is $\Omega(1)$, i.e., bounded from below by some positive constant. Hence the objective value (the maximum ratio) is $\Omega(1)$, too.

For some $\delta > 0$ that will be specified later, we round the values $x_i \leq 1$ to integer multiples of δ, and we round the values $x_i > 1$ to powers of $1 + \delta$. Next we decide on the entire kernel, i.e., we select pieces to form the bags with more than two pieces, in all possible ways.

By standard calculations, there exist only $\Theta(1) \log t \cdot \delta^{-1}$ different discrete values. Since pieces with the same rounded size are not distinguished, there exist no more than $(\Theta(1) \log t \cdot \delta^{-1})^{4t}$ different choices of a kernel: To see this, let us describe the kernel as a sequence of bags separated by "end of bag" symbols, and every bag as a sequence of pieces. The exponent $4t$ accounts for the at most $3t$ pieces and at most t "end of bag" symbols.

For every possible kernel we pair up the remaining pieces in an optimal way (the largest with the smallest, etc.), in $O(n)$ time, and we compute the maximum ratio of all bags. Eventually we pick the minimum solution. Note that we need the sorted sequence of sizes for the pairing, but actually we only have to count the number of pieces with each rounded size, and the rounded sizes are sorted in advance by bucketsorting.

Due to the lower bound $\Omega(1)$ it suffices to choose some suitable $\delta := \Theta(\varepsilon)$. Then, after applying another constant scaling factor, Lemma 12 ensures a relative error of at most the desired ε. □

MAX–MIN RATIO can be treated similarly but needs a different scaling of the discrete values. The following FPTAS uses a coarse 4-approximation to calibrate the rounding.

Theorem 20 MAX–MIN RATIO *with* $\varphi = 1$ *and* $t \leq n/3$ *can be solved within a factor* $1 - \varepsilon$ *of optimum in* $(\Theta(1) \log t \cdot \varepsilon^{-1})^{4t} \cdot O(n)$ *time.*

Proof Suppose that we know already some interval $[u, v]$ that contains the objective value of the given instance. We mark all pieces larger than some $\Theta(tv)$ as "large".

Since the number of pieces in every bag is limited by $O(t)$, large pieces cannot belong to bags that attain the minimum ratio. That is, the exact sizes of these pieces are not relevant.

For all other pieces not marked as 'large", we round the values $x_i \leq v$ to integer multiples of some δv, and we round the values $x_i > v$ to powers of $1 + \delta v$.

Due to the bounded size $\Theta(tv)$, there exist no more than $\Theta(1) \log t \cdot (\delta v)^{-1}$ different discrete values.

The rest works as in Theorem 19, except that "minimum" and "maximum" are swapped.

Since u bounds the objective value from below, it suffices to choose some $\delta := \Theta(\varepsilon u/v)$ to obtain the desired ε. This finally results in the time bound $(\Theta(v/u) \log t \cdot \varepsilon^{-1})^{4t} \cdot O(n)$.

It remains to specify the requested interval $[u, v]$. The following paragraph is not part of the algorithm description, but it only proves that $u = x_b/2$ and $v = 2x_b$ are suitable, which also means $v/u = O(1)$.

By Lemma 1, some upper bound on the objective value of MAX–MIN RATIO with $\varphi = 0$ is given by x_b, where $b = (n - t)/2$ is the number of bags. With $\varphi = 1$, the objective can increase by a factor at most 2. Therefore $v = 2x_b$ is a valid upper bound. We generate a trivial solution as follows. We put each of the b largest pieces into an own bag, and we add at least one and at most two smaller pieces to each bag. (Since $t \leq n/3$, we have $b \geq n/3$, and $b \leq n/2$ holds, too.) Clearly, the ratio in every bag is at least $x_b/2$. Therefore $u = x_b/2$ is a valid lower bound. □

7 Further Research

We list various questions that are either unsolved or go beyond the scope of this paper.

One direction is to modify the problem definitions in order to pay attention to further aspects. For instance, a solution to MAX–MIN SPLITTING may be perceived as unfair, as some agents get much more than the guaranteed minimum, because the entire good must be divided. (In particular, the sum of the F smallest sizes of pieces is a trivial upper bound on the objective value.) To circumvent these issues we may relax the problem and aim at giving maximal but equal amounts to all agents, possibly leaving the remainder of goods unused, or allow some agents to exceed the fragmentation F. Yet another question is to what extent a "combined" problem, aiming at a minimum ratio of the largest and smallest share, behaves differently than MIN–MAX SPLITTING and MAX–MIN SPLITTING separately.

We have concentrated on $F = 1$ and the "graph-theoretic" case $F = 2$ which is already subtle. It may be possible to generalize several results to any fixed $F > 2$.

The non-uniqueness of solutions to PERFECT SPLITTING with $F = 2$ suggests further extensions of this problem. For instance, let us equip the vertex set with pairwise distances (spatial distances, dissimilarity of tasks, etc.). Then we may prefer perfect solutions where also some distance measure is minimized, such as the maximum or the sum of the lengths of the chosen edges.

It remains open whether the exact MIN–MAX (MAX–MIN) AVERAGE problems with $\varphi = 1$ are FPT in the parameter $t = 2m - n$. There is an essential difference to PERFECT SPLITTING: The proof of Theorem 9 shows that we can simply cut off the pairs with sum 1 first. This part cannot be straightforwardly generalized to MIN–MAX (MAX–MIN) SPLITTING. Since the sums can deviate from 1, it is not clear which

pieces we should pair up. Our results on small t are only parameterized approximation schemes. Approximation allowed us to round the values, such that the number of essentially different pieces to choose from (for the kernel) does not depend on n. Actually we conjecture $W[1]$-hardness due to relationship with the k-SUM problem [1], but we could not establish a reduction from that problem either.

In order to extend the efficiently solvable cases discovered before, we have focused on small b or t. But we have not considered the worst-case approximation ratios that can be achieved in polynomial time when m ranges over the whole interval $[n/2, n-2]$. Some more efficient PTAS may be possible, too.

Finally, solutions from any algorithms may be further improved by a local search post-processing, e.g., by pairwise exchange operations of pieces between the bags. For our particular problems it might be useful to characterize solutions that are local optima with respect to such exchange steps.

Acknowledgements Open access funding provided by Chalmers University of Technology. The author would like to thank the anonymous reviewers for their advice and very careful reading.

References

1. Abboud, A., Lewi, K., Williams, R.: On the parameterized complexity of k-SUM. CoRR arXiv:abs/1311.3054 (2013)
2. Aumann, Y., Dombb, Y.: The efficiency of fair division with connected pieces. In: Saberi, A. (ed.) WINE, LNCS 6484, pp. 26–37 (2010)
3. Babel, L., Kellerer, H., Kotov, V.: The k-partitioning problem. Math. Methods OR **47**, 59–82 (1998)
4. Blum, M., Floyd, R.W., Pratt, V.R., Rivest, R.L., Tarjan, R.E.: Time bounds for selection. J. Comput. Syst. Sci. **7**, 448–461 (1973)
5. Cechlárová, K., Pillárová, E.: On the computability of equitable divisions. Discrete Optim. **9**, 249–257 (2012)
6. Chen, L., Jansen, K., Luo, W., Zhang, G.: An efficient PTAS for parallel machine scheduling with capacity constraints. In: Chan, T.H.H., Li, M., Wang, L. (eds.), COCOA, LNCS 10043, pp. 608–623 (2016)
7. Edmonds, J., Pruhs, K.: Cake cutting really is not a piece of cake. ACM Trans. Alg. 7, article 51 (2011)
8. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, Dallas (1979)
9. Gellert, L., Sanyal, R.: On degree sequences of undirected, directed, and bidirected graphs. Eur. J. Comb. **64**, 113–124 (2017)
10. Golovach, P.A., Mertzios, G.B.: Graph editing to a given degree sequence. Theor. Comput. Sci. **665**, 1–12 (2017)
11. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems: theoretical and practical results. J. ACM **34**, 144–162 (1987)
12. Hochbaum, D.S., Shmoys, D.B.: A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. SIAM J. Comput. **17**, 539–551 (1988)
13. Horowitz, E., Sahni, S.: Exact and approximate algorithms for scheduling nonidentical processors. J. ACM **23**, 317–327 (1976)
14. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problems. J. ACM **22**, 463–468 (1975)

15. Kellerer, H., Kotov, V.: A 3/2-approximation algorithm for k_i-partitioning. Oper. Res. Lett. **39**, 359–362 (2011)
16. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer, Berlin (2004)
17. Martinovic, J., Jorswieck, E., Scheithauer, G.: The skiving stock problem and its application to cognitive radio networks. IFAC-PapersOnLine **49**, 99–104 (2016)
18. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, Oxford (2006)
19. Procaccia, A.D.: Cake-cutting algorithms. In: Handbook of Computational Social Choice. Cambridge University Press, pp. 261–283 (2015)
20. Reitzig, R., Wild, S.: Building Fences Straight and High: An Optimal Algorithm for Finding the Maximum Length You Can Cut k Times from Given Sticks. Algorithmica **80**, 3365–3396 (2018)