Dividing Splittable Goods Evenly and With Limited Fragmentation

Peter Damaschke[1]

1 **Department of Computer Science and Engineering**
Chalmers University, 41296 Göteborg, Sweden
ptr@chalmers.se

Abstract

A splittable good provided in n pieces shall be divided as evenly as possible among m agents, where every agent can take shares of at most F pieces. We call F the fragmentation. For $F = 1$ we can solve the max-min and min-max problems in linear time. The case $F = 2$ has neat formulations and structural characterizations in terms of weighted graphs. Here 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. Moreover, case $F = 2$ is fixed-parameter tractable in the parameter $2m - n$. The results also give rise to various open problems.

1998 ACM Subject Classification G.2.2 Graph Theory, I.1.2 Algorithms

Keywords and phrases packing, load balancing, weighted graph, linear-time algorithm, parameterized algorithm

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.9

1 Introduction

Suppose that we are given n pieces of a good, and these pieces have sizes $x_1, \ldots, x_n$. The good shall be divided among m agents, thereby respecting certain 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, we have $y_j = \sum_{i=1}^{n} z_{ij}$ for all agents j, and $\sum_{j=1}^{m} z_{ij} \leq x_i$ for all pieces i. All mentioned variables are non-negative. The agents are identical, and so are the pieces of the good (apart from their different sizes).

Ideally we would like to divide the good evenly and completely, that is: $y_1 = \ldots = y_m$ 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 by the solution, but their number is limited by F.) However, every piece may be divided among an unlimited number of agents.

One possible motivation is that pieces of land, at n different locations and with areas $x_1, \ldots, x_n$, shall be assigned to farmers in a fair way. Besides fairness, 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. 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. An example where these assumptions are realistic is grading of the n exercises of an exam by m graders.

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 [6]. An action equivalent to cutting is packing items into containers, as in the Bin Packing and Knapsack problems [5]. (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, and to our best knowledge their complexities have not been studied before. 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.

The paper is organized as follows. In Section 2 we define splitting problems where the good shall be divided completely and the minimum share shall be maximized, or vice versa. We call a solution perfect if all shares are equal. In Section 3 we solve the case of fragmentation $F = 1$ in linear time. However, we first derive weaker $O(n \log n)$ time bounds, because this makes the presentation easier, and the $O(n)$ solution relies on the (barely practical) linear-time selection but should still be of theoretical interest. In Section 4 we are concerned with the case of fragmentation $F = 2$. Here we focus on structural properties of the problem of getting perfect solutions, but this may also serve as a basis for, e.g., approximation algorithms for the more general optimization versions. We describe solutions in a natural way by weighted graphs. Then we present a linear-time elimination algorithm that finds a tree-like perfect solution whenever $m \geq n - 1$. Based on the graph-theoretic structure we also prove strong NP-completeness of the general problem, and NP-completeness already for the special case when m is slightly smaller than $n - 1$. For m being close to the smallest possible value $n/2$ we show fixed-parameter tractability. In Section 5 we point out possible directions of further research.

2 Preliminaries

For clarity we provide formal definitions of the problems. We always silently presume that F is a fixed positive integer called the *fragmentation*. Common to all problems are the input and some of the constraints:

Splitting:
Given are n positive numbers $x_1, \ldots, x_n$.
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.

To fully specify the actual problems we only mention these additional constraints, 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*. In such a solution, all agents get the same amount, and no goods are held back.

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 are 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 Min-Max problem is 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, as large as possible. However, a solution may be perceived as unfair, in the sense that other agents get significantly more because the entire good must be divided. Also, the size of the smallest piece is a trivial upper bound on the objective value. To circumvent these issues one may define modified Max-Min problems which aim at giving maximal but equal amounts to all agents, possibly leaving the remainder of goods unused. One could also relax the condition on fragmentation F and allow some outliers. However, we cannot study all variants in this paper, and we focus on the basic problems.

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 $F = 1$.

As a trivial observation, either of the first two problems always permits an optimal solution where, for every piece i, all $z_{ij} > 0$ are equal. 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 $y_j = z_{ij} = x_i/p_i$. Due to this observation on the possible y_j-values we define the following sequence Y.

▶ **Definition 1.** Y is 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 given x_i. But the efficient computation of exact optimal solutions is a bit less obvious. We discuss this matter in the following subsections.

3.1 Maximizing the minimum

The following lemma deals with the simple problem being inverse to Max-Min Splitting.

▶ **Lemma 2.** *For any fixed y with $0 < y \leq \min_i x_i$, let k be the maximum number of agents*[1] *such that every agent can obtain an amount at least y. This number k satisfies:*
(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 split 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.

[1] As y is fixed, for notational convenience we do not mention the argument y and call this number only k.

(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 seen 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 yields a characterization of the optimal solution.

▶ **Lemma 3.** *The optimal objective value for* Max-Min Splitting *with* $F = 1$ *is*

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

where the maximum is taken over all solutions.

Proof. First suppose that $Y[m] < \min_i x_i$.

Then we can apply Lemma 2 to $y := Y[m]$. The maximum number k of agents that can be served is, due to (b), equal to 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 2 to y'. The maximum number k' of agents that can be served is, due to (b), equal to 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.

The other case to consider is $Y[m] \geq \min_i x_i$. Now we can apply Lemma 2 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 due to the problem specification, $\min_i x_i$ is also a trivial upper bound on the objective value, which finally proves the assertion also in this case. ◀

The previous lemmas yield already an algorithm for Max-Min Splitting: First, it is trivial to figure out whether m agents can get a share of at least $\min_i x_i$ each. If so, then this solution is optimal, and we are done. If not, then $Y[m] < \min_i x_i$. In this case, find the value $y := Y[m]$ and then determine the p_i as in Lemma 2 (a), using this y.

However, in order to save time we want to get the value $Y[m]$ without naively following Definition 1 and generating all $m - 1$ preceding elements of Y. The intuition for a faster approach is to search the optimal value near the average. In the following we can always suppose $Y[m] < \min_i x_i$.

▶ **Lemma 4.** *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 3 yields $\bar{y} \geq Y[m]$, and Lemma 2 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 and fractional part $p_i := \lfloor q_i \rfloor$ and $r_i := q_i - \lfloor q_i \rfloor$, respectively. Lemma 2 (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. We adopt the unit cost measure where comparisons and algebraic operations with real numbers take constant time.

▶ **Theorem 5.** Max-Min Splitting *with* $F = 1$ *can be solved in* $O(n \log n)$ *time.*

Proof. First we compute $\bar{y}$ and all p_i (as defined in Lemma 2 with $y := \bar{y}$ and x_i/p_i), as well as k and $Y[k] = \min_i x_i/p_i$. The latter equation holds due to Lemma 2 (c). These calculations cost $O(n)$ time.

By Lemma 3, the optimal value is $Y[m]$. In the following we determine the value $Y[m]$. Note that k and $Y[k]$ are known from the calculations above, and $Y[k] \geq Y[m]$ and $m-k \leq n$ hold due to Lemma 4.

If $m \leq k$, then $Y[m] = Y[k]$, and we are done with this part. 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. Now we describe a possible way to identify $Y[k], \ldots, Y[m]$.

Let us sort the n ratios x_i/p_i in decreasing order and call this sequence R. Since $Y[k] = \min_i x_i/p_i$, the last element of R is exactly $Y[k]$. We call it the *marked* element, for later reference. In R we store not only the values x_i/p_i but also x_i and p_i separately. We move a pointer in R from left to right. For every ratio encountered in R we compute the ratio with incremented denominator ($p_i := p_i + 1$) and insert it at the correct position in R. Note that this position is always to the right of the pointer; in particular, the new ratio may become the new rightmost element of R. This step is repeated until the pointer reaches a position $m-k$ elements to the right of the marked element, and then we stop.

The analysis is simple: Since new ratios are always inserted to the right of the pointer, and R comprises *all* elements of Y between the marked $Y[k]$ and the pointer, it follows that we are at $Y[m]$ when we stop, and we can output its value. In order to support fast insertion of a new ratio into R, we also host R in a balanced search tree. Therefore this procedure can be done in $O(n \log n)$ time.

For every i we recompute p_i by $p_i := \lfloor x_i/Y[m] \rfloor$, and the amount given to the p_i agents assigned to the ith piece is x_i/p_i. ◀

Some comments on the time bound are in order. Theorem 5 actually says that we need $O(n \log n)$ time to compute the numbers of agents assigned to each piece and their shares, in an optimal solution. Under the unit cost measure, this time bound is independent of m which may be arbitrarily larger than n. The "physical" division, i.e., assigning positive values to m variables z_{ij}, takes $O(m)$ additional time in a trivial postprocessing phase.

Modifications of the algorithm can also solve a variant of Max-Min Splitting with $F = 1$ where not all goods need to be distributed. Such an algorithm would not get to the pieces being smaller than the objective value.

3.2 Minimizing the maximum

In order to minimize $\max_j y_j$ we use the sequence Y from Definition 1 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. Similarly as before, we start from the simple problem being inverse to Min-Max Splitting.

▶ **Lemma 6.** *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 $\lfloor x_i/y \rfloor$ 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. ◀

Again we get a simple characterization of the optimal solution.

▶ **Lemma 7.** *The optimal objective value for* Min-Max Splitting *with* $F = 1$ *is*

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

where the maximum is taken over all solutions.

Proof. We apply Lemma 6 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. The minimum number k of agents needed is, due to (b), equal to 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 6 to y'. The minimum number k' of agents needed is, due to (b), equal to 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 $Y[m - n + 1]$. ◀

The proof of Theorem 5 showed already that we can determine $Y[m]$ from $\bar{y}$ in $O(n \log n)$ time. By a similar procedure we can also determine $Y[m - n + 1]$, which is optimal for Min-Max Splitting due to Lemma 7. (Note that n is known from the instance.) Again this costs only $O(n \log n)$ time, and the same remarks as earlier apply to the actual construction of the solution. We can readily state:

▶ **Theorem 8.** Min-Max Splitting *with* $F = 1$ *can be solved in* $O(n \log n)$ *time.*

3.3 Splitting in linear time

An obvious question is whether $O(n \log n)$ time is actually needed. Re-inspecting the proof of Theorem 5) we see: The non-trivial case is $r := m - k > 0$ (but $r \leq n$), and there we only need to find the rth largest ratio after the marked element $Y[k]$. Thus it may not be necessary to keep our sequence R of ratios sorted. But a difficulty is that the sought element is not simply the rth largest $x_i/(p_i + 1)$, where the p_i denote the initial values obtained from $\bar{y}$. Rather, several ratios x_i/p with the same index i but varying p may be larger than the sought element. It is not immediately clear in which order we should generate new ratios and do comparisons.

But, in fact, we can achieve $O(n)$ time. We present this improvement here separately, as it does not look very practical (see below). Yet the optimal time bound may be of interest. First we need a separation property of our sequence of ratios:

▶ **Lemma 9.** *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. ◀

Now we outline a linear-time algorithm for Max-Min Splitting, improving upon the implementation details in Theorem 5. First, a few preparations and definitions are needed. We find the maximum x_i in $O(n)$ time. Without loss of generality let $x_1 = \max_i x_i$. Considering the sequence of ratios x_1/q with integers q, to the right of the marked element $Y[k]$, we call every interval $(x_1/q, x_1/(q+1)]$ a *bin*. Note that every such interval includes its right endpoint. For a set S of ratios, the *generation step* is the following procedure. For every ratio x_i/p_i in S, we increment the denominator by 1 and compute the number q of the bin that contains $x_i/(p_i+1)$. Using divisions this requires only $O(1)$ time per element. The Selection problem is to find the tth smallest number in an unsorted set, for a prescribed number t.

First we apply the generation step to all initial values x_i/p_i (that is, with the p_i obtained from $\bar{y}$, as earlier). Then we scan the bins from left to right, i.e., for increasing q. For every q we take all ratios that are currently in the qth bin and apply the generation step to them. At the same time we count the total number of ratios seen so far, including the x_1/q. As soon as this number reaches r, we know that the sought ratio is in the current bin. Lemma 9 ensures that every bin contains at most one ratio x_i/p (p integer) for every index i, thus $O(n)$ ratios altogether. Let $t := r - s$, where s is the number of ratios in all previous bins. Hence we can finally apply an $O(n)$ time Selection algorithm [1] to the current bin, in order to find the sought ratio at the tth position in this bin. For the closely related Min-Max Splitting problem we can proceed similarly. This shows:

▶ **Theorem 10.** Max-Min Splitting *and* Min-Max Splitting *with $F = 1$ can be solved in $O(n)$ time.*

A caveat is that $O(n)$-time Selection algorithms suffer from a large hidden constant in the time bound. Instead of a deterministic algorithm we may use the randomized Quickselect, but then we only get $O(n)$ expected time. One may wonder if $O(n)$ worst-case time can be accomplished without invoking Selection. However this is not possible. For instance, if all x_i are close to each other and n does not divide m, then finding $Y[m]$ is equivalent to finding the (m mod n)th largest x_i.

Figuratively speaking, our splitting problems are "Selection-complete" under "simple" linear-time reductions. This statement could be formalized in an algebraic model of computation that cares about constant factors (e.g., by counting the exact depth in a decision tree model). In a more general perspective it may be interesting to – in this way – strictly classify problems that are all linear-time solvable but of different complexity when it comes to constant factors.

4 Perfect Solutions for Fragmentation F=2

In the following we study the case $F = 2$. It is natural to represent any instance of a splitting problem, along with its solution, as a weighted graph:

▶ **Definition 11.** The *solution graph* of a solution to a splitting problem with $F = 2$ is a 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}. Every y_j is the sum of the weights of the two ports of the jth edge. We consider y_j as the weight 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.

In this section we want to characterize which instances, given by n positive vertex weights $x_1, \ldots, x_n$ and an integer m, allow a perfect solution (see Section 2). Without loss of generality we can assume $\sum_{i=1}^{n} x_i = m$, hence a perfect solution must satisfy $y_j = 1$ for all j. That is, all edge weights are 1.

4.1 Many agents make it easy

▶ **Theorem 12.** *Every instance of* Perfect Splitting *with* $F = 2$ *and* $m \geq n - 1$ *has a solution whose solution graph is a tree, possibly with loops attached to some vertices. Moreover, such a solution can be computed in* $O(n)$ *time.*

Proof. We classify the vertices in several categories depending on their weights x_i. Vertex i is called large if $x_i > 1$, normal if $x_i = 1$, medium if $1/2 < x_i < 1$, and small if $0 < x_i \leq 1/2$.

Our strategy works as follows. We create certain edges of weight 1, reduce the remaining weights of the incident vertices accordingly (and obeying some simple rules), and recursively solve the residual instances. We only need to show that the process terminates only when the vertex weights are zero, and it can be implemented in $O(n)$ time. In detail:

First suppose that $m > n$. Then there exists a large vertex i. Hence we can attach a loop to it and update $x_i := x_i - 1$ and $m := m - 1$, while n is unchanged. By an inductive argument, we can attach $m - n$ loops to large vertices, until $m = n$ is reached. The number of loops attached to every large vertex can be decided, e.g., in a greedy fashion: Choose any large vertex i and create $\lceil x_i \rceil - 1$ loops, take another large vertex, and so on, but stop as soon as the total number of loops reaches $m - n$. The computations obviously require only $O(n)$ arithmetic operations.

Next suppose that $m = n$. If all vertices are normal, then we append another loop to each vertex, and we are done. If not all vertices are normal, then there still exists some large vertex, and we attach another loop to it, as above. Thus we reach $m = n - 1$.

From now on suppose that $m = n - 1$. Assume that we find two vertices i and j such that $x_i + x_j > 1$ and $x_j < 1$. Then we join these vertices by an edge of weight 1, which is divided as follows. The port at vertex j gets the weight x_j, and the port at vertex 1 gets the rest $1 - x_j$. Accordingly we update the vertex weights to $x_j := 0$ and $x_i := x_i - 1 + x_j > 0$. This means, there remain $n := n - 1$ vertices with positive weights, and $m := m - 1$ edges to fix. Hence the invariant $m = n - 1$ is preserved in the residual instance, and we can iterate this procedure as long as possible.

It remains to prove the existence of such a pair of vertices i and j satisfying $x_i + x_j > 1$ and $x_j < 1$, and to find some efficiently. Since $m < n$, at least one medium or small vertex

exists. As long as some large or normal vertex exists, too, we obviously get a pair as desired. Now suppose that all vertices are medium or small. We can also take a pair of medium vertices, if it exists. Thus suppose in the sequel that all vertices are small, except at most one medium vertex. Now the equation $m = n - 1$ restricts the possibilities as follows. If two small vertices exist, then these are the only vertices: We have $n = 2$ and $x_1 = x_2 = 1/2$, hence we can join the two vertices by a final edge. If only one vertex is small, then there exists exactly one other vertex which is medium. Hence we still have $n = 2$ and $x_1 + x_2 = 1$, and we can insert a final edge. The case that no vertex is small is impossible, since then $n = 1$ and $m = x_1 > 0$, a contradiction. Altogether this shows that we always find two desired vertices until $n = 2$ is reached, and then we can finish up the solution.

The above proof does not only show the existence of a perfect solution whenever $m \geq n-1$, but it describes already a simple elimination algorithm that computes a perfect solution. In each step, the updates take $O(1)$ time. The next edge is always built from two vertices from certain categories (large, normal, medium, small). Since arbitrary vertices from the respective categories can be chosen, it suffices to maintain four unsorted sets of vertices, hence we can also update these sets and pick the needed vertices in $O(1)$ time. Therefore the overall time is $O(n)$.

Consider the graph of non-loop edges inserted by the algorithm until any moment. Upon insertion of every edge, the weights of its vertices were positive, and the weight of exactly one of them drops to zero. This implies the invariant that every connected component of the graph retains exactly one vertex with positive weight. From this it follows, furthermore, that every new edge merges two connected components. Thus the final solution graph is a tree, possibly with additional loops. ◀

Similarly to the time bounds for $F = 1$, we have not counted in the time for the trivial postprocessing that actually splits the pieces: Constructing the $O(m)$ loops costs $O(m)$ additional time. But the solution graph, i.e., the tree and the numbers of loops at its vertices, are computed in $O(n)$ time.

It is apparent from the algorithm that the tree, and thus the solution, is in general not unique. This gives rise to interesting additional problems, also in view of the motivations in Section 1. For instance, assuming that the vertices have pairwise distances (spatial distances, dissimilarity of tasks, etc.) we may prefer perfect solutions where also some distance measure (maximum, total, weighted, etc.) is minimized for the chosen edges.

4.2 Structural characterization and hardness

Apart from the last remark, Theorem 12 completely settles the case $m \geq n - 1$. In the following we also allow $m < n$ (but $m \geq n/2$, since otherwise not all goods can be divided with fragmentation $F = 2$). The conditions in Theorem 12 suggest the following definition.

▶ **Definition 13.** Let V be a set of elements called vertices and indexed by $1, \ldots, n$. Every vertex i has a weight 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 $\sum_{i \in I} x_i \geq |I| - 1$.

In fact, the existence of perfect solutions can now be characterized as follows.

▶ **Theorem 14.** *An instance $(x_1, \ldots, x_n)$ of* Perfect Splitting *with $F = 2$, where $\sum_{i=1}^{n} x_i = m$ (and m is 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 11, with vertex set V and with m edges (some of which may be loops). Let $C(k)$ denote the kth connected component of G, where the indexing is arbitrary. Let n_k and m_k denote the number of vertices and edges, respectively, in $C(k)$. Since $\sum_{i=1}^{n} x_i = m$, every agent gets an amount of 1. Hence, for every k, the vertex set V_k of C_k is integral. Specifically, the sum of vertex weights in $C(k)$ equals m_k which is trivially an integer. 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 denotes their total weight. For every V_k we consider an instance of Perfect Splitting with the given vertex weights and m_k agents. Due to $m_k \geq n_k - 1$ and Theorem 12, this instance has a solution with m_k agents. Since $m = \sum_k m_k$, the solutions to all these k instances together form a solution of the entire instance. ◀

While Perfect Splitting with $F = 2$ is easy for $m \geq n - 1$ due to Theorem 12, the complexity jumps for $m < n - 1$. The reason is that, unfortunately, it is hard to find a partitioning as required in Theorem 14, as we show next. At first glance hardness might appear counterintuitive because with fragmentation $F = 2$ it should always be possible, within an elimination process as in Theorem 12, to take amounts missing to some $z_{ij} = 1$ from some other piece. But the catch is that all remaining pieces might be too small, and then the fragmentation $F = 2$ is not sufficient. Anyway, by a reduction from 3-Partition (which is a natural candidate that has been reduced earlier to similar packing and scheduling problems [2]) we can show:

▶ **Theorem 15.** Perfect Splitting *with* $F = 2$ *is strongly NP-complete, and so are* Max-Min Splitting *and* Min-Max Splitting.

Proof. We give a polynomial-time reduction from the strongly NP-complete 3-Partition problem to Perfect Splitting. A triplet is a set of exactly three elements. An instance P of 3-Partition consists of $3k$ positive rational numbers that shall be partitioned into k triplets such that the sum of the numbers in each triplet is the same. The instance is a multiset, i.e., the $3k$ numbers are not necessarily distinct. Without loss of generality let their sum be k, hence the sum in each triplet must be 1. Thus we can further assume $x \leq 1$ for all x in P, otherwise P has, trivially, no solution. We also fix some small number $d > 0$, in fact, any number 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 x from P into $2(1/3 + dx)/(1 + d)$. 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 14, Q has a perfect solution.

Conversely, suppose that Q has a perfect solution. Using Theorem 14 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 $x \leq 1$ for all x 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, the sum in each triple 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. ◀

Usual NP-hardness holds already when m is slightly smaller than $n-1$ (giving a clear dichotomy together with Theorem 12), and the proof is less technical:

▶ **Theorem 16.** Perfect Splitting *with $F = 2$ and $m = n - t$ is NP-complete for every fixed $t \geq 2$, and so are* Max-Min Splitting *and* Min-Max Splitting.

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 14, 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 [2]. An instance of Subset Sum consists of positive rational numbers $y_1 \ldots, y_n$, and the goal is to find a subset I of indices such that $\sum_{i\in I} y_i$ equals a prescribed value 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 $\sum_{k=1}^{n} y_k = 2$, such that we have to divide the sum into $1+1$. Now we can also assume $y_k < 1$ for all k, otherwise the instance has, trivially, no solution. Finally, we simply set $x_k := 1 - y_k$ for all k. The equivalence to Perfect Splitting with $F = 2$ and $m = n - 2$ is evident.

For $t > 2$ we finally add $2(t-2)$ further items $x_i = 1/2$ to the instance. Arguments are similar; note that the additional items must form $t - 2$ pairs, in a partitioning into heavy integral sets. ◀

4.3 Few agents make it easy, too

The reductions showing hardness led to instances with $m/n \geq 2/3$. The problem becomes "easy" if m is close to the smallest possible value $n/2$. (Readers not being familiar with fixed-parameter tractability are referred to a textbook like [7].)

▶ **Theorem 17.** Perfect Splitting *with $F = 2$ is fixed-parameter tractable (FPT) in the parameter $t := 2m - n$.*

Proof. Consider graphs with n vertices and m edges. We refer to connected components with two and three vertices as pairs and triplets, respectively. As a preparatory consideration we construct extremal graphs where the number q of vertices not being in pairs is maximized. We can assume that some pairs exist, since otherwise $q = n$ is, trivially, maximal.

If some connected component C is not a tree, then we can remove an edge such that C remains connected. We use this edge to join some pair to another component. This strictly increases q. Hence assume that all connected components are trees. If some tree has 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, which strictly increases q again. Hence, if q is maximized, then all connected components are pairs and triples (unless $q = n$). With p pairs and t' triples we have $m = p + 2t'$ and $n = 2p + 3t'$,

hence $t = 2m - n = t'$. Since the maximum q is shown to be $q = 3t' = 3t$, we get that at most $3t$ vertices are not in pairs.

Now consider any instance of Perfect Splitting with $F = 2$, and any two indices i and j with $x_i + x_j = 1$. Assume that the instance has a solution. We consider the partitioning specified in Theorem 14 and refer to its heavy integral sets as bags.

Assume that i and j are in different bags, say, in $I \cup \{i\}$ and $J \cup \{j\}$. We rearrange them to new bags $\{i, j\}$ and $I \cup J$. The pair is obviously a heavy integral set. Since the original bags were integral and the weight $x_i + x_j = 1$ has been removed, also $I \cup J$ is integral. Since the original bags were heavy, $I \cup \{i\}$ and $J \cup \{j\}$ have sums at least $|I|$ and $|J|$, respectively. Hence $I \cup J$ has a sum at least $|I| + |J| - 1$, which means that it is heavy. This shows, under the above assumption, 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, $K \cup \{i, j\}$ with $K \neq \emptyset$ is a bag. We split it in two bags K and $\{i, j\}$. Clearly, K is integral. Since $K \cup \{i, j\}$ was heavy, it has a sum at least $|K| + 1$, such that at least a sum $|K|$ 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.

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. (This is the "data reduction" phase, in the terminology of FPT algorithms.) Then we solve the residual instance, that is, we search for a partitioning as in Theorem 14, of the remaining indices. The given instance has a perfect solution if and only if the residual instance has.

The pairing phase is correct 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. By traversing a sorted list of the weights simultaneously in ascending and descending order, this phase is easily implemented in $O(n \log n)$ time. The residual instance has a size at most $3t$, and we may solve it naively. ◀

By way of contrast, Theorem 16 excludes an FPT (even an XP) algorithm in the parameter $t = n - m$ (unless P=NP).

5 Further Research

By Theorem 15, the optimization versions of or splitting problems with $F = 2$ (and $n > m$) are strongly NP-complete and therefore do not allow FPTAS (unless P=NP). On the other hand, the structural result in Theorem 14 together with the algorithms in Theorem 12 suggests that one should try and partition the set of pieces into $n - m$ heavy subsets (bags) and then assign agents to them such that the average amounts per agent in the bag are as balanced as possible. The latter step would work as in case $F = 1$, treating the bags as pieces. In general, the bags will not be integral, but the smaller the fractional parts of the sums are, the better the solutions should be. Minimizing the fractional parts roughly resembles the minimum makespan scheduling problem (although the objectives are also quite different). The latter problem 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 $n - m$ of bags) is fixed; see [3, 4]. Altogether, we conjecture that similar approximation schemes can be obtained for our splitting problems. Another conjecture is that our FPT result for small $2m - n$ extends to the optimization versions.

Other open questions were mentioned in the technical sections. Furthermore, as we have seen, the "graph-theoretic" case $F = 2$ is already subtle, but several results may be generalized to $F > 2$.

References

1 Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973. URL: https://doi.org/10.1016/S0022-0000(73)80033-9, doi:10.1016/S0022-0000(73)80033-9.

2 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

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

4 Ellis Horowitz and Sartaj Sahni. Exact and approximate algorithms for scheduling non-identical processors. *J. ACM*, 23(2):317–327, 1976. URL: http://doi.acm.org/10.1145/321941.321951, doi:10.1145/321941.321951.

5 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004.

6 John Martinovic and Guntram Scheithauer. Integer rounding and modified integer rounding for the skiving stock problem. *Discrete Optimization*, 21:118–130, 2016. URL: https://doi.org/10.1016/j.disopt.2016.06.004, doi:10.1016/j.disopt.2016.06.004.

7 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Univ. Press, 2006.