Fair Division with Splitting

Samuel Bismuth, Vladislav Makarov, Erel Segal-Halevi and Dana Shapira

Abstract

We consider a variant of the n-way number partitioning problem, in which some fixed number s of items can be split between two or more bins. We show a two-way polynomial-time reduction between this variant and a second variant, in which the maximum bin sum must be within a pre-specified interval. We prove that, for any fixed integer $n \geq 3$, the second variant can be solved in polynomial time if the length of the allowed interval is at least $(n-2)/n$ times the maximum item size, and it is NP-complete otherwise. Using the equivalence between the variants, we prove that, for any fixed integer $n \geq 3$, n-way number-partitioning with s split items can be solved in polynomial time if $s \geq n-2$, and it is NP-complete otherwise. Using the same reduction, we provide empirical evidence that the number of split items for a perfect partition in practice is much lower than the worst-case scenario.

1 Introduction

In the classic n-way number partitioning problem (where $n \geq 2$ is a fixed integer), the input is a list $\mathcal{X} = (x_1, \ldots, x_m)$ of m non-negative integers, and the objective is to find an n-way partition (a partition into n bins) of $\mathcal{X}$ such that the maximum bin sum is minimized. Given an n-way partition, we denote by b_i the sum of item sizes in bin i.

> $n-$Way$-$MinMax$(\mathcal{X})$: *Minimize* $\max(b_1, \ldots, b_n)$*, where* $b_1, \ldots, b_n$ *are sums of bins in an* n*-way partition of* $\mathcal{X}$*.*

This abstract problem has many concrete applications. One of them is the famous Makespan Minimization problem: given a fixed number, n, of identical machines, and a list of m jobs with given processing times, assign each job to a machine, such that the maximum completion time is minimized. Another application is fair allocation of indivisible chores (items with negative values).

The problem is known to be NP-hard for every fixed $n \geq 2$ [8]. One could consider a continuous (fractional) variant of $n-$Way$-$MinMax, in which each item can be split between two or more bins. This variant is very easy to solve: by splitting each item equally into n parts and putting a part in each bin, it is possible to attain a *perfect partition*, that is, a partition with $b_1 = \cdots = b_n = \frac{1}{n}\sum_i x_i$.

While theoretical problems are usually classified as either "discrete" or "continuous", practical partitioning problems may lie in between these two extremes: every item can be split if needed, but the splitting may be expensive or inconvenient, and therefore the number of split items should be bounded. As an example, consider $n = 2$ heirs who inherited $m = 3$ houses and have to divide them fairly. The house values are $\mathcal{X} = (100, 200, 400)$. If all houses are considered discrete, then an equal division is not possible. If all houses can be split, then an equal division is easy to attain by giving each heir 50% of every house, but it is inconvenient since it requires all houses to be jointly managed. A natural solution that can be used in practice is to decide in advance that *a single* house can be split.[1] In this case, after receiving the input, we can determine that splitting the house with value 400 lets us attain a division in which each heir receives the same value of 350.[2]

[1]We have heard from real-estate assessors (Achikam Bitan, Amnon Nizri) that the solution of splitting a small number of houses is often used in practical estate-division cases.

[2] Split the house worth 400 such that one heir gets 7/8 of it, and the other gets 1/8 of it plus both the

Motivated by these practical examples in fair division and machine scheduling (see Appendix D for more examples), we define a variant of the n−Way−MinMax problem, which accepts, in addition to the list $\mathcal{X}$ of integers to partition, a parameter $s \in \mathbb{N}$, $0 \leq s \leq m$, which specifies that at most s items from $\mathcal{X}$ are allowed to be split between bins.

> n−s−Split−MinMax*($\mathcal{X}$): Minimize* $\max(b_1, \ldots, b_n)$*, where* $b_1, \ldots, b_n$ *are sums of bins in an n-way partition of $\mathcal{X}$ in which at most s items are split.*

We denote by n−s−Split−Perfect the decision version of n−s−Split−MinMax, which only asks if there exists a perfect partition (a partition with equal bin sums). Note that the problem definition does not determine in advance which s items will be split, but only bounds their number. The solver may decide which items to split after receiving the input.

When $s = 0$, n−s−Split−MinMax is equivalent to the NP-hard n−Way−MinMax. In contrast, when $s \geq n-1$ the problem is easily solvable by the following algorithm: put the items on a line, cut the line into n pieces with an equal total value, and put each piece in a bin. Since $n-1$ cuts are made, at most $n-1$ items need to be split. So for $n = 2$, the runtime complexity of the n−s−Split−MinMax problem is well-understood (assuming $\mathsf{P} \neq \mathsf{NP}$): it is polynomial-time solvable if and only if $s \geq 1$. The main goal of this paper is to analyze the runtime complexity of the problem for any fixed integers $n \geq 3$ and $1 \leq s \leq n-2$, as a function of m — the number of input integers, and $\log_2 S$ — where S is defined as the average bin size, $S := (\sum_{i=1}^{m} x_i)/n$. Note that $\log_2 S$ is upper bounded by the number of bits in the input. Our main results are:

Theorem 1.1. *For any fixed integers $n \geq 3$ and $s \geq n-2$,* n−s−Split−MinMax *can be solved in time $O(poly(m, \log S))$.*

Theorem 1.2. *For any fixed integers $n \geq 3$ and $s < n-2$,* n−s−Split−MinMax *is* NP*-hard. Moreover, even its decision version* n−s−Split−Perfect*, which asks if there exists a perfect partition (a partition with equal bin sums), is* NP*-complete.*

In order to solve n−s−Split−MinMax, we consider another variant of n-way number partitioning, a decision problem which asks whether the maximum bin sum is within a pre-specified interval around the average bin sum, $S := (\sum_i x_i)/n$. Note that S is the *optimal fractional solution*, that is, the solution that can be attained if all items can be split. The runtime complexity of this problem depends on the size of the allowed interval. In general, the problem is NP-complete when the interval is "small" and in P when the interval is "large". Specifically, when $n = 2$, the runtime complexity depends on the ratio of the allowed interval to the *average bin sum*, while when $n \geq 3$ it depends on the ratio of the allowed interval to the *largest item size*. Therefore, we present two variants of the problem. The first is parametrized by a rational number $t \geq 0$:

> n−t−Interval*($\mathcal{X}$): Decide if there exists a partition of $\mathcal{X}$ into n bins with sums $b_1, \ldots, b_n$ such that* $S \leq \max(b_1, \ldots, b_n) \leq (1+t) \cdot S$*, where* $S := (\sum_i x_i)/n$*.*

The second is parametrized by a rational number $d \geq 0$:

> n−d−Interval*($\mathcal{X}$): Decide if there exists a partition of $\mathcal{X}$ into n bins with sums $b_1, \ldots, b_n$ such that* $S \leq \max(b_1, \ldots, b_n) \leq S + d \cdot M$*, where* $S := (\sum_i x_i)/n$ *and* $M := (\max_i x_i)/n$*.*

Note that the lower bound $S \leq \max(b_1, \ldots, b_n)$ always holds by the pigeonhole principle, so we could have named our problems "Upper Bound" instead of "Interval". We prefer to use "interval" because our proofs are based on both lower bounds and upper bounds. Our secondary results are:

100 and 200 houses.

Theorem 1.3. *For $n = 2$ and any rational number $t > 0$, the n−t−*Interval*($\mathcal{X}$) problem can be solved in time $O(\text{poly}(m, \log S, 1/t))$.*

Theorem 1.4. *For any fixed integer $n \geq 3$ and rational number $d \geq n-2$, the n−d−*Interval *($\mathcal{X}$) problem can be solved in time $O(\text{poly}(m, \log S))$.*

Our algorithms use as building-blocks several FPTAS algorithms, which we define next.

Theorem 1.5. *For any fixed $n \geq 3$ and rational number $d < n - 2$, the n−d−*Interval*($\mathcal{X}$) problem is NP-complete.*

Definition 1.1. *An* FPTAS *for n−*Way−MinMax *is an algorithm that finds, for each real $\epsilon > 0$, an n-way partition of $\mathcal{X}$ with $OPT \leq \max(b_1, \ldots, b_n) \leq (1 + \epsilon) \cdot OPT$, where OPT is the smallest possible value of $\max(b_1, \ldots, b_n)$ in the given instance, in time $O(poly$ $(m, 1/\epsilon, \log S))$.*

Notice that the n−t−Interval problem can be seen as a slightly strengthened version of an FPTAS, where the relative deviation from optimality depends on the optimal fractional solution, instead of the optimal integral solution. Despite the similarity, the problems are not identical. While n−Way−MinMax has an FPTAS [23], it does not automatically solve n−t−Interval. In Section 3.1 we show how an FPTAS for a slightly different problem can be used to solve 2−t−Interval and prove Theorem 1.3. In the rest of Section 3 we analyze the n−d−Interval problem and prove Theorem 1.4 and Theorem 1.5.

In Section 4, we prove a two-way polynomial-time reduction between the partitioning with interval target and partitioning with split items problems. Using this reduction and Theorems 1.4 and 1.5, we conclude the proof of Theorems 1.1 and 1.2.

In Appendix F, using the same reduction, we develop a practical (not polynomial-time) algorithm for solving n−s−Split−MinMax for any $s \geq 0$. The algorithm can use any practical algorithm for solving the n−Way−MinMax problem; we use a variant of the Complete Greedy Algorithm [11]. We apply this algorithm to various randomly generated instances, and analyze the effect of s on the quality of the attained solution. We find out that, in most cases, much fewer than $n - 1$ splits are sufficient for attaining a perfect partition. For example, in over 74% of the settings we checked, $n/2$ splits are sufficient for a perfect partition.

In the supplement, we present two complementary results.

In Appendix B, we show that assuming n is fixed is necessary for our positive results: when n is part of the input (not fixed in advance), both d−Interval($\mathcal{X}, n$) and s−Split−Perfect ($\mathcal{X}, n$) are strongly NP-hard for every fixed rational number d and integer s.

In Appendix C, we analyze a different way to control the amount of splitting. Instead of considering the number of split items, one can measure the number of times each item is split. The number of splittings is at least the number of split items, but it might be larger. For example, a single item split into 10 different bins counts as 9 splittings. We name this variant n−s−Times−Split−Perfect($\mathcal{X}$). It is similar to the n−s−Split−Perfect($\mathcal{X}$) problem with the exception that s denotes the number of splittings. When $s \geq n - 1$ a perfect partition can be easily found by the same "line-cutting" algorithm. We prove that this variant is NP-hard for all $s < n - 1$, thus completing the classification of its run-time complexity.

2 Related work

The idea of finding fair allocations with a bounded number of split items originated from Brams and Taylor [3, 4]. They presented the *Adjusted Winner* (AW) procedure for allocating items among two agents with possibly different valuations. AW finds an allocation that is *envy-free* (no agent prefers the bundle of another agent), *equitable* (both agents receive the same subjective value), and *Pareto-optimal* (there is no other allocation where some agent gains and no agent loses), and in addition, at most a single item is split between the agents. Hence, AW solves a problem that is similar to $2-1-\mathsf{Split}-\mathsf{Perfect}(\mathcal{X})$ but more general, since AW allows the agents to have different valuations to the same items. AW was applied (at least theoretically) to division problems in divorce cases and international disputes [5, 14] and was studied empirically in [18, 7]. The AW procedure is designed for two agents. For $n \geq 3$ agents, the number of split items allowed was studied in an unpublished manuscript of Wilson [22] using linear programming techniques. He proved the existence of an *egalitarian* allocation of goods (i.e., an allocation in which all agents have a largest possible equal utility [16]), with at most $n-1$ split items; this can be seen as a generalization of $n-(n-1)-\mathsf{Split}$ $-\mathsf{Perfect}(\mathcal{X})$.

Goldberg et al. [9] studied the problem of *consensus partitioning.* In this problem, there are n agents with different valuations, and the goal is to partition a list of items into some k subsets (where k and n may be different), such that the value of each subset is $1/k$ for each agent. They prove that a consensus partitioning with at most $n(k-1)$ split items can be found in polynomial time.

Most similar to our paper is the recent work of Sandormirskiy and Segal-Halevi [17]. Their goal is to find an allocation among n agents with different valuations, which is both fair and *fractionally Pareto-optimal (*fPO), a property stronger than Pareto-optimality (there is no other discrete or fractional allocation where some agent gains and no agent loses). This is a very strong requirement: when n is fixed, and the valuations are *generic* (i.e., for every two agents, no two items have the same value-ratio), the number of fPO allocations is polynomial in m, and it is possible to enumerate all such allocations in polynomial time. Based on this observation, they present an algorithm that finds an allocation with the smallest number of split items, among all allocations that are fair and fPO. In contrast, in our paper, we do not require fPO, which may allow allocations with fewer split items. However, the number of potential allocations becomes exponential, so enumerating them all is no longer feasible.

Recently, Bei et al. [1] studied an allocation problem where some items are divisible and some are indivisible. In contrast to our setting, in their setting the distinction between divisible and indivisible items is given in advance, that is, the algorithm can only divide items that are pre-determined as divisible. In our setting, only the *number* of divisible items is given in advance, but the algorithm is free to choose *which* items to split after receiving the input.

There are various works on combinatorial problems such as some variants of bin packing or knapsack problems involving for example some penalties for each split-items or including additional cost for fragmented items [2] [15] [13] [10].

Fractional relaxations of integer linear programs are very common in approximation algorithm design. However, most works do not explicitly bound the number of fractional variables. One similar notion is related to *preemption* in job-scheduling. In the world of machine scheduling, where the problem $n-\mathsf{Way}-\mathsf{MinMax}$ is known as "makespan-minimization on identical machines".

Fractional scheduling is often done using *preemption.* Preemption means that a job can be split into parts, each of which can be scheduled independently, either on the same or on a different machine, with the additional constraint that no two parts of a job are scheduled on different machines at the same time. We found two works in which the amount of preemption is bounded: Soper and Strusevich [21] allow at most one preemption, while Liu and Cheng [12] consider a variant in which there is a penalty for each preemption.

3 Partition with Interval Target

In this section we analyze the two related problems n−t−Interval and n−d−Interval. Recall that, by the definition of these problems, they accept as input a list $\mathcal{X}$ of positive integers with sum $n \cdot S$ and maximum element $n \cdot M$, where S and M are rational numbers.

3.1 The n−t−Interval problem

Given an instance of n−t−Interval($\mathcal{X}$), we say that a partition of $\mathcal{X}$ is *t-feasible* if $S \leq \max(b_1, \ldots, b_n) \leq (1+t) \cdot S$, where $b_1, \ldots, b_n$ are the bin sums. The n−t−Interval($\mathcal{X}$) problem is to decide whether a t-feasible allocation exists.

Note that, when $t \geq 1$, the problem can be decided easily by a linear-time greedy algorithm (see Appendix H). Therefore, we focus below on the case $t < 1$.

As a first (incomplete) attempt to solve n−t−Interval, let us apply a known FPTAS for the n−Way−MinMax problem [23]. Denote the largest bin sum in the solution obtained by the FPTAS for a given list $\mathcal{X}$ and a given $\epsilon > 0$ by FPTAS(n−Way−MinMax($\mathcal{X}$), ϵ).

If FPTAS(n−Way−MinMax($\mathcal{X}$), ϵ)$\leq (1+t) \cdot S$, then the partition returned by the FPTAS is t-feasible, so the answer to n−t−Interval($\mathcal{X}$) is "yes".

If FPTAS(n−Way−MinMax($\mathcal{X}$), ϵ)$> (1+t) \cdot S$, then the partition returned by the FPTAS is not t-feasible, but we *cannot* conclude that the answer to n−t−Interval($\mathcal{X}$) is "no", since there may be some other t-feasible partition. However, this outcome does give us valuable information about the instance: by the definition of FPTAS, it implies that the optimal value to n−Way−MinMax($\mathcal{X}$) is larger than $(1+t) \cdot S/(1+\epsilon)$. This implies that, in *any* n-way partition of $\mathcal{X}$, there is at least one bin with sum larger than $(1+t) \cdot S/(1+\epsilon)$.

Definition 3.1. *Given an instance of* n−t−Interval*($\mathcal{X}$), a real number $\epsilon > 0$, and a partition of $\mathcal{X}$ into n bins, an* almost-full bin *is a bin with sum larger than* $(1+t) \cdot S/(1+\epsilon)$.

In the previous paragraph we have proved the following lemma:

Lemma 3.1. *For any integer $n \geq 2$, rational $t > 0$ and real $\epsilon > 0$, if* FPTAS(n−Way −MinMax($\mathcal{X}$), ϵ) $> (1+t) \cdot S$*, then in any n-way partition of $\mathcal{X}$, at least one bin is almost-full.*

To gain more information on the instance, we apply an FPTAS for a constrained variant of n−Way−MinMax, with a *Critical Coordinate (CC)*. For an integer $n \geq 2$, a list $\mathcal{X}$, and a rational number $t > 0$, we define the following problem:

> n−t−Way-MinMax−CC*($\mathcal{X}$): Minimize* $\max(b_2, \ldots, b_n)$ *subject to* $b_1 \leq (1+t) \cdot S$*, where $b_1, \ldots, b_n$ are sums of bins in an n-way partition of $\mathcal{X}$.*

The general technique developed by Woeginger [23] for converting a dynamic program to an FPTAS can be used to design an FPTAS for n−t−Way-MinMax−CC; we give the details in Appendix E.1. We denote by FPTAS(n−t−Way-MinMax−CC($\mathcal{X}, t$), ϵ) the largest bin sum in the obtained solution. The following lemma is the key for our algorithms.

Lemma 3.2. *For any $n \geq 2, t > 0, \epsilon > 0$, if* FPTAS($n$−$t$−Way-MinMax−CC($\mathcal{X}$), ϵ) $>$ $(1+t) \cdot S$*, then in any t-feasible n-way partition of $\mathcal{X}$, at least* two *bins are almost-full.*

Proof. Suppose by contradiction that there exists a t-feasible partition of $\mathcal{X}$ with at most one almost-full bin. It is possible to reorder the bins in the partition such that bin 1 has the largest sum; the resulting partition is still t-feasible, and still has at most one almost-full bin. Since bin 1 has the largest sum, if there is one almost-full bin, it must be bin 1. So in any case, bins $2, \ldots, n$ are not almost-full, so $\max(b_2, \ldots, b_n) \leq (1+t) \cdot S/(1+\epsilon)$. Moreover, $b_1 \leq (1+t) \cdot S$ since the partition is t-feasible. Therefore, FPTAS(n−t−Way-MinMax−CC ($\mathcal{X}$), ϵ) $\leq (1+t) \cdot S$ by the definition of FPTAS. This contradicts the lemma assumption. □

Using Lemma 3.2, we can now derive a complete algorithm for $2-t-$Interval.

Algorithm 1 $2-t-$Interval$(\mathcal{X})$

```
1: $b_2 \longleftarrow$ FPTAS($2-t-$Way-MinMax$-$CC($\mathcal{X}$), $\epsilon = t/2$).
2: If $b_2 \leq (1+t) \cdot S$, return "yes".
3: Else, return "no".
```

Theorem 3.1 (Theorem 1.3)**.** *For any rational $t > 0$, Algorithm 1 solves the $2-t-$Interval $(\mathcal{X})$ Problem in time $O(poly(m, \log S, 1/t))$, where m is the number of items in $\mathcal{X}$ and S is the average bin size.*

Proof. The run-time of Algorithm 1 is dominated by the run-time of the FPTAS for the problem $2-t-$Way-MinMax$-$CC, which is $O(\text{poly}(m, \log S, 1/\epsilon)) = O(\text{poly}(m, \log S, 1/t))$ (we show in Appendix E.3.1 that the exact run-time is $O(\frac{m}{t} \log S)$). It remains to prove that Algorithm 1 indeed solves $2-t-$Interval correctly. If b_2, the returned bin sum of FPTAS($2-t-$Way-MinMax$-$CC($\mathcal{X}$), $\epsilon = t/2$), is at most $(1+t) \cdot S$, then the partition found by the FPTAS is t-feasible, so Algorithm 1 answers "yes" correctly.

Otherwise, by Lemma 3.2, in any t-feasible partition of $\mathcal{X}$ into two bins, both bins are almost-full. This means that, in any t-feasible partition, both b_1 and b_2 are larger than $(1+t) \cdot S/(1+\epsilon)$, which is larger than S since $\epsilon = t/2$. This means that $b_1 + b_2 > 2S$. But this is impossible, since the sum of all items is $n \cdot S = 2S$ by assumption. Therefore, no t-feasible partition exists, and Algorithm 1 answers "no" correctly. □

For the following sections, we will need a variant of $n-t-$Interval with an additional constraint, that all bins must have an equal number of items.

> $n-t-$Interval$-$Eq*($\mathcal{X}$): Decide if there exists a t-feasible partition of $\mathcal{X} = (x_1, \ldots, x_m)$ into n bins such that each bin contains exactly m/n items.*

Clearly, if m is not a multiple of n then the answer is "no"; the interesting case is that m is a multiple of n.

We can solve $n-t-$Interval$-$Eq for $n = 2$ in a similar way to Algorithm 1. This requires an FPTAS to the corresponding equal-cardinality variant of $2-t-$Way-MinMax$-$CC, which we call $2-t-$Way-MinMax$-$Eq$-$CC. An FPTAS for this problem can be designed by the technique of Woeginger [23]; the details are in Appendix E.2. By plugging this FPTAS into Algorithm 1, we get an algorithm for solving $2-t-$Interval$-$Eq:

Theorem 3.2. *There is an algorithm for solving $2-t-$Interval$-$Eq$(\mathcal{X})$ for any rational $t > 0$ in time $O(poly(m, 1/t, \log S))$, where m is the number of input items in $\mathcal{X}$ and S is the average bin size.*

Remark 3.1. *The reader may wonder why we cannot use a similar algorithm for $n \geq 3$. For example, we could have considered a variant of $n-t-$Way-MinMax$-$CC with two critical coordinates:*

> *Minimize* $\max(b_3, \ldots, b_n)$ *subject to* $b_1 \leq (1+t) \cdot S$ *and* $b_2 \leq (1+t) \cdot S$*, where* $b_1, b_2, b_3, \ldots, b_n$ *are sums of bins in an n-way partition of $\mathcal{X}$.*

If the FPTAS *for this problem does not find a t-feasible partition, then any t-feasible partition must have at least three almost-full bins. Since not all bins can be almost-full, one could have concluded that there is no t-feasible partition into $n = 3$ bins.*

Unfortunately, the problem with two critical coordinates probably does not have an FPTAS *even for $n = 3$, since it is equivalent to the* Multiple Subset Sum *problem, which does not have an* FPTAS *unless* P=NP *[6]. In the next subsection we handle the case $n \geq 3$ in a different way.*

3.2 The $n-d-\mathsf{Interval}$ problem: an algorithm for $n \geq 3$ and $d \geq n-2$

Given an instance of $n-d-\mathsf{Interval}(\mathcal{X})$, where the sum of items is $n \cdot S$ and the maximum item is $n \cdot M$, where $S, M \in \mathbb{Q}$, we say that a partition of $\mathcal{X}$ is *d-possible* if $S \leq \max(b_1, \ldots, b_n) \leq S + d \cdot M$, where $b_1, \ldots, b_n$ are the bin sums. The $n-d-\mathsf{Interval}(\mathcal{X})$ problem is to decide whether a d-possible allocation exists.[3] Given an instance of $n-d-\mathsf{Interval}(\mathcal{X})$, we let $t := dM/S$, so that a partition is d-possible if-and-only-if it is t-feasible.

The algorithm starts by running $\mathsf{FPTAS}(n-t-\mathsf{Way\text{-}MinMax}-\mathsf{CC}(\mathcal{X}), \epsilon = t/4m^2)$. If the FPTAS finds a t-feasible partition, we return "yes". Otherwise, by Lemma 3.2, any t-feasible partition must have at least two almost-full bins.

Now, we take a detour from the algorithm and prove some existential results about partitions with two or more almost-full bins.

We assume that there are more items than bins, that is, $m > n$. This assumption is without loss of generality, since if $m \leq n$ the problem is trivial.

3.2.1 Structure of partitions with two or more almost-full bins

We distinguish between big, medium and small items defined as follows. A *big-item* is an item with size greater than $nS(\frac{t}{n-2} - 2\epsilon)$, while a *small-item* is an item with size smaller than $2nS\epsilon$. All other items are *medium-items.* Our main structural Lemma is the following.

Lemma 3.3. *Suppose that $d \geq n-2$, $t = dM/S < 1$, $\epsilon = t/4m^2$ and the following properties hold.*

(1) There is no t-feasible partition with at most 1 *almost-full bin;*

(2) There is a t-feasible partition with at least 2 *almost-full bins.*

Then, there is a t-feasible partition with the following properties.

(a) Exactly two bins (w.l.o.g. bins 1 and 2) are almost-full.

(b) The sum of every not-almost-full bin $i \in \{3, \ldots, n\}$ satisfies

$$\left(1 - \frac{2}{n-2}t - 2\epsilon\right) \cdot S \leq b_i \leq \left(1 - \frac{2}{n-2}t + (n-1)2\epsilon\right) \cdot S.$$

(c) Every item in an almost-full bin is a big-item.

(d) Every item in a not-almost-full bin is either a small-item, or a big-item larger or equal to every item in bins 1,2.

(e) There are no medium-items *at all.*

(f) Every not-almost-full bin contains the same number of big-items, say ℓ, where ℓ is an integer (it may contain, in addition, any number of small-items).

(g) Every almost-full bin contains $\ell + 1$ big-items (and no small-items).

As an example to this situation, consider an instance with 7 items, all of which have size 1, with $n = 5$ and $d = 3$. Then, there is a d-possible partition with two almost-full bins: $(1, 1), (1, 1), (1), (1), (1)$, and no d-possible partition with 1 or 0 almost-full bins. See Appendix I for details.

A full proof of the lemma appears in Appendix J; here we provide a sketch.

[3]Note that when $d = n-1$, the answer to the problem is always yes. We can prove this by iterating over the items, assigning each item to the bin with the smallest sum. Recall that the biggest item is equal to nM, so by the pigeonhole principle, if the current item size is nx, then the smallest bin sum before the addition is at most $S - x$, so the new bin sum is at most $S + (n-1)x$. By assumption, $x \leq M$, so all bin sums are at most $S + (n-1)M$. Therefore, the interesting case is $d < n-1$.

Proof Sketch. We start with an arbitrary t-feasible partition with some $r \geq 2$ almost-full bins $1, \ldots, r$, and convert it using a sequence of transformations to another t-feasible partition satisfying properties (a)–(g), as explained below. Note that the transformations are not part of our algorithm and are only used to prove the lemma. First, we note that there must be at least one bin which is not almost-full, since the sum of an almost-full bin is larger than S whereas the sum of all n bins is $n \cdot S$.

For (a), if there are $r \geq 3$ almost-full bins, we move any item from one of the almost-full bins $3, \ldots, r$ to some not-almost-full bin. We prove that, as long as $r \geq 3$, the target bin remains not-almost-full. This transformation is repeated until $r = 2$ and only bins 1 and 2 remain almost-full.

For (b), for the lower bound, if there is $i \in \{3, \ldots, n\}$ for which b_i is smaller than the lower bound, we move an item from bins $1, 2$ to bin i. We prove that bin i remains not-almost-full, so by assumption (1), bins 1, 2 must remain almost-full. We repeat until b_i satisfies the lower bound. Once all bins satisfy the lower bound, we prove that the upper bound is satisfied too.

For (c), if bin 1 or 2 contains an item that is not big, we move it to some bin $i \in \{3, \ldots, n\}$. We prove that bin i remains not-almost-full, so by assumption (1), bins 1, 2 must remain almost-full. We repeat until bins 1 and 2 contain only big-items.

For (d), if some bin $i \in \{3, \ldots, n\}$ contains an item bigger than $2nS\epsilon$ and smaller than any item in bin 1 or bin 2, we exchange it with an item from bin 1 or 2. We prove that, after the exchange, b_i remains not-almost-full, so bins 1, 2 must remain almost-full. We repeat until bins 1,2 contain only the smallest big-items. Note that transformations (b), (c), (d) increase the sum in the not-almost-full bins $3, \ldots, n$, so the process must end.

For (e), it follows logically from properties (d) and (c): if bins 1,2 contain only big items and the other bins contain only big and small items, then the instance cannot contain any medium items (that are neither big nor small). For clarity and verification, we provide a stand-alone proof.

For (f), we use the fact that the difference between two not-almost-full bins is at most $2nS\epsilon$ by property (b), and show that it is too small to allow a difference of a whole big-item.

For (g), because by (d) bins 1 and 2 contain the smallest big-items, whereas their sum is larger than bins $3, \ldots, n$, they must contain at least $\ell + 1$ big-items. We prove that, if they contain $\ell + 2$ big-items, then their sum is larger than $(1 + t)S$, which contradicts t-feasibility. □

Let $B \subseteq \mathcal{X}$ be the set of big items in $\mathcal{X}$. Properties (f) and (g) imply:

Corollary 3.1. *Suppose that $d \geq n - 2$, $t = dM/S$ and $\epsilon = t/4m^2$. If there is a t-feasible partition with at least two almost-full bins, and no t-feasible partition with at most one almost-full bin, then $|B| = n\ell + 2$ for some positive integer ℓ.*

3.2.2 Back to the algorithm

We have left the algorithm at the point when $\mathsf{FPTAS}(n\text{–}t\text{–}\mathsf{Way}\text{–}\mathsf{MinMax}\text{–}\mathsf{CC}(\mathcal{X}), \epsilon = t/4m^2)$ did not find a t-feasible partition. Lemma 3.2 implies that if a t-feasible partition exists, then there exists a t-feasible partition satisfying all properties of Lemma 3.3 and Corollary 3.1. We can find such a partition (if it exists) in two steps:

- **For bins 1,2:** Find a t-feasible partition of the $2\ell + 2$ smallest items in B into two bins with $\ell + 1$ items in each bin.

- **For bins** $3, \ldots, n$**:** Find a t-feasible partition of the remaining items in $\mathcal{X}$ into $n - 2$ bins.

For bins $3, \ldots, n$, we use the FPTAS for the problem $(n-2)$−Way−MinMax. If it returns a t-feasible partition, we are done. Otherwise, by Lemma 3.1, every partition into $(n-2)$ bins must have at least one almost-full bin. But by Lemma 3.3(a), all bins $3, \ldots, n$ are not almost-full — a contradiction. Therefore, if the FPTAS does not find a t-feasible partition, we answer "no".

Bins 1 and 2 require a more complicated algorithm. [4] We use the following notation (where n denotes the same parameter $n \geq 3$ of the original n−d−Interval problem):

- $B_{1:2}$ is the set of $2\ell + 2$ smallest items in B (where B is the set of big items in $\mathcal{X}$).
- $2S_{1:2}$ is the sum of items in $B_{1:2}$ (for some $S_{1:2} \in \mathbb{Q}$).
- $nM_{1:2}$ is the largest item in $B_{1:2}$ (for some $M_{1:2} \in \mathbb{Q}$).

Construct a new set, $\overline{B_{1:2}}$, by replacing each item $x \in B_{1:2}$ by its "inverse", defined by $\overline{x} := nM_{1:2} - x$. Since all items in $B_{1:2}$ are big-items, all inverses are between 0 and $2nS\epsilon$ (see the proof of Lemma 3.3(f) in Appendix J). Denote the sum of inverses by $2\overline{S_{1:2}}$.

Given a t-feasible partition of $B_{1:2}$ with sums b_1 and b_2 and $\ell+1$ items in each bin, denote the sums of the corresponding partition of $\overline{B_{1:2}}$ by $\overline{b_1}$ and $\overline{b_2}$, respectively. Since both bins contain $\ell + 1$ items, $\overline{b_i} = (\ell+1) \cdot nM_{1:2} - b_i$, for $i \in \{1, 2\}$, and $\overline{S_{1:2}} = (\ell+1) \cdot nM_{1:2} - S_{1:2}$. Now,

$$\begin{aligned} b_1 \leq (1+t)S \iff & \overline{b_1} \geq (\ell+1)nM_{1:2} - (1+t)S \\ \iff & \overline{b_2} \leq 2\overline{S_{1:2}} - (\ell+1)nM_{1:2} + (1+t)S \\ & = \overline{S_{1:2}} + \big(\overline{S_{1:2}} - (\ell+1)nM_{1:2}\big) + (S + tS) \\ & = \overline{S_{1:2}} + (S + tS - S_{1:2}), \end{aligned}$$

and similarly, $\overline{b_1} \leq \overline{S_{1:2}} + (S + tS - S_{1:2})$.

So the problem of finding a t-feasible partition of $B_{1:2}$ with $\ell + 1$ items in each bin is equivalent to the problem of finding a $\overline{t}$-feasible partition of $\overline{B_{1:2}}$ with $\ell + 1$ items in each bin, where $\overline{t} := (S + tS - S_{1:2})/\overline{S_{1:2}}$. By Theorem 3.2, this problem can be solved in $O(\text{poly}(m, 1/\overline{t}, \log S))$ time.

It remains to prove that $1/\overline{t}$ is polynomial in m. By Lemma 3.3(d), in each of the not-almost-full bins, there are ℓ items that are at least as large as $nM_{1:2}$ (in addition to some small-items). Therefore, $nS \geq 2S_{1:2} + (n-2) \cdot \ell \cdot nM_{1:2}$, which implies that

$$S - S_{1:2} \geq -\frac{n-2}{n} S_{1:2} + \frac{n-2}{n} \ell \cdot nM_{1:2}.$$

Since $d \geq n - 2$, $tS = dM \geq (n-2)M \geq (n-2)M_{1:2}$.

Summing up these two inequalities gives:

$$\begin{aligned} S - S_{1:2} + tS & \geq \frac{n-2}{n} \Big(\ell \cdot nM_{1:2} - S_{1:2} + nM_{1:2} \Big) \\ & = \frac{n-2}{n} \Big((\ell+1) \cdot nM_{1:2} - S_{1:2} \Big) = \frac{n-2}{n} \cdot \overline{S_{1:2}}. \end{aligned}$$

Therefore, $\overline{t} \geq \frac{n-2}{n}$, so $1/\overline{t} \in O(1)$ and the sub-problem can be decided in time $O(\text{poly}(m, \log S))$.

[4] Using a simple FPTAS for Subset Sum to solve our problem is not possible since the total size of the items may equal $2(S + dM)$ and our goal is to divide the items among two bins of capacity $S + dM$, which means that no approximation is allowed. Therefore, we need to pre-process the items in some way. The technique we use is to inverse the items. Even after inverting the items, we need an additional cardinality constraint enforcing that there are exactly $\ell + 1$ items in each bin.

3.2.3 Complete algorithm

We are now ready to present the complete algorithm for n–d–Interval, presented in Algorithm 2.

Algorithm 2 n–d–Interval (complete algorithm)

```
1: $t \longleftarrow dM/S$ and $\epsilon \longleftarrow t/(4m^2)$.
2: If FPTAS($n$–$t$–Way-MinMax–CC($\mathcal{X}$), $\epsilon$) $\leq (1+t)\cdot S$, return "yes".
3: $B \longleftarrow \{x_i \in \mathcal{X} \mid x_i > nS(\frac{t}{n-2} - 2\epsilon)\}$        ▷ big items
4: If $|B|$ is not of the form $n\ell + 2$ for some integer $\ell$, return "no".
5: $B_{1:2} \longleftarrow$ the $2\ell + 2$ smallest items in $B$.        ▷ break ties arbitrarily
6: $B_{3..n} \longleftarrow \mathcal{X} \setminus B_{1:2}$.        ▷ big and small items
7: $(b_3, \ldots, b_n) \longleftarrow$ FPTAS(($n-2$)–Way–MinMax($B_{3..n}$), $\epsilon$) ▷ an $(n-2)$–way partition of $B_{3..n}$
8: If $\max(b_3, \ldots, b_n) > (1+t)S$, return "no".
9: $\overline{B_{1:2}} \longleftarrow \{nM_{1:2} - x \mid x \in B_{1:2}\}$ and $\bar{t} \longleftarrow (S + tS - S_{1:2})/\overline{S_{1:2}}$.
10: Look for a $\bar{t}$-feasible partition of $\overline{B_{1:2}}$ into two subsets of $\ell + 1$ items as explained in Theorem 3.2.
11: If a $\bar{t}$-feasible partition is found, return "yes". Else, return "no".
```

Theorem 3.3 (Theorem 1.4)**.** *For any fixed integer $n \geq 3$ and rational number $d \geq n-2$, Algorithm 2 solves n–d–Interval($\mathcal{X}$) in $O(poly(m, \log S))$ time, where m is the number of items in $\mathcal{X}$, and S is the average bin size.*

Proof. If Algorithm 2 answers "yes", then clearly a t-feasible partition exists. To complete the correctness proof, we have to show that the opposite is true as well.

Suppose there exists a t-feasible partition. If the partition has at most one almost-full bin, then by Lemma 3.2, it is found by the FPTAS in step 2. Otherwise, the partition must have at least two almost-full bins, and there exists a t-feasible partition satisfying the properties of Lemma 3.3.

By Corollary 3.1, the algorithm does not return "no" in step 4. By properties (a) and (b), there exists a partition of $B_{3..n}$ into $n-2$ bins $3, \ldots, n$ which are not almost-full. By Lemma 3.1, the FPTAS in step 7 finds a partition with $\max(b_3, \ldots, b_n) \leq (1+t)S$. The final steps, regarding the partition of $B_{1:2}$, are justified by the discussion at Section 3.2.2. The complete running time $O(\text{poly}(m, \log S))$ of Algorithm 2 is justified by the running time of the FPTAS for n–t–Way-MinMax–CC and for 2–t–Way-MinMax–Eq–CC. Note that $1/t$ is polynomial in m since $1/t = S/dM \leq mM/dM = m/d = O(m)$ since d is fixed. The exact running time, $O(m^4 \log S)$, is detailed in Appendix E.3.2. □

4 Partition with Split Items

To solve the optimization problem n–s–Split–MinMax, we present the following decision problem. For a fixed number $2 \leq n \in \mathbb{N}$ of bins, given a list $\mathcal{X}$, the number of split items $0 \leq s \leq m$, $s \in \mathbb{N}$, and a rational number $t \geq 0$, define:

> n–s–t–Split–Bound*($\mathcal{X}$): Decide if there exists a partition of $\mathcal{X}$ into n bins, with at most s split items, such that* $\max(b_1, \ldots, b_n) \leq (1+t)S$.

The special case $t = 0$ corresponds to the n–s–Split–Perfect($\mathcal{X}$) problem. The following Lemma shows that, w.l.o.g., we can consider only the largest items for splitting.

Lemma 4.1. *For every partition with $s \in \mathbb{N}$ split items and bin sums $b_1, \ldots, b_n$, there exists a partition with the same bin sums $b_1, \ldots, b_n$ in which only the s* largest *items are split.*

Proof. Consider a partition in which some item x is split between two or more bins, whereas some item $y > x$ is allocated entirely to some bin i. Construct a new partition as follows:

– Move item x entirely to bin i;

– Split item y into two parts with sizes x and $y - x$;

– Remove the part with size x from bin i, and split it among the bins in the same proportions that the item x was split originally.

All bin sums remain the same. Repeat the argument until only the largest items are split. $\square$

Theorem 4.1. *For any fixed integers $n \geq 2$ and $d \geq 0$, there is a polynomial-time reduction from n–d–Interval($\mathcal{X}$) to n–d–0–Split–Bound($\mathcal{X}$) (that is, n–s–t–Split–Bound($\mathcal{X}$) with $s = d$ and $t = 0$).*

Proof. Given an instance $\mathcal{X}$ of n–d–Interval($\mathcal{X}$), construct an instance $\mathcal{X}'$ of n–d–0–Split –Bound($\mathcal{X}'$) by adding d items of size nM, where nM is the size of the biggest item in $\mathcal{X}$.

First, assume that $\mathcal{X}$ has a d-possible partition. Then there are n bins with a sum at most $S + dM$. Take the d added items of size nM and add them to the bins, possibly splitting some items between bins, such that the sum of each bin becomes exactly $S + dM$. This is possible because the sum of the items in $\mathcal{X}'$ is $nS + dnM = n(S + dM)$. The result is a 0-feasible partition of $\mathcal{X}'$ with at most d split items.

Second, assume that $\mathcal{X}'$ has a 0-feasible partition with at most d split items. Then there are n bins with sum exactly $S + dM$. By Lemma 4.1, we can assume the split items are the largest ones, which are the d added items of size nM. Remove these items to get a partition of $\mathcal{X}$. The sum in each bin is now at most $S + dM$, so the partition is d-possible.

This construction is done in polynomial time, which completes the proof. $\square$

Corollary 4.1 (Theorem 1.2)**.** *For a fixed integers $n \geq 3$ and $s \in \{0, 1, \ldots, n-3\}$, the problem n–s–Split–MinMax is NP-hard, and n–s–0–Split–Bound($\mathcal{X}$) $\equiv$ n–s–Split–Perfect ($\mathcal{X}$) is NP-complete.*

Proof. Theorem A.1 and Theorem 4.1 together imply that both problems are NP-hard. The problem n–s–Split–Perfect($\mathcal{X}$) is in NP since given a partition, summing the sizes of the elements (or element fractions) in each bin lets us to check in linear time whether the partition has equal bin sums. $\square$

Theorem 4.2. *For any fixed integers $n \geq 2, s \geq 0$ and rational $t \geq 0$, there is a polynomial-time reduction from n–s–t–Split–Bound($\mathcal{X}$), to n–d–Interval($\mathcal{X}$) for some rational $d \geq s$.*

Proof. Given an instance $\mathcal{X}$ of n–s–t–Split–Bound($\mathcal{X}$), denote the sum of all items in $\mathcal{X}$ by nS and the largest item size by nM where $S, M \in \mathbb{Q}$. Construct an instance $\mathcal{X}'$ of n–d–Interval($\mathcal{X}'$) by removing the s largest items from $\mathcal{X}$. Denote the sum of remaining items by nS' for some $S' \leq S$, and the largest remaining item size by nM' for some $M' \leq M$. Note that the size of every removed item is between nM' and nM, so $sM' \leq S - S' \leq sM$. Set $d := (S + tS - S')/M'$, so $S' + dM' = S + tS$. Note that $d \geq (S - S')/M' \geq s$.

First, assume that $\mathcal{X}$ has a t-feasible partition with s split items. By Lemma 4.1, we can assume that only the s largest items are split. Therefore, removing the s largest items results in a partition of $\mathcal{X}'$ with no split items, where the sum in each bin is at most $S + tS = S' + dM'$. This is a d-possible partition of $\mathcal{X}'$.

Second, assume that $\mathcal{X}'$ has a d-possible partition. In this partition, each bin sum is at most $S' + dM' = S + tS$, so it is a t-feasible partition of $\mathcal{X}'$. To get a t-feasible partition of $\mathcal{X}$, take the s previously-removed items and add them to the bins, possibly splitting some items between bins, such that the sum in each bin remains at most $S + tS$. This is possible because the sum of all items is $nS \leq n(S + tS)$.

This construction is done in polynomial time, which completes the proof. $\square$

Combining Theorem 4.2 with Theorem 3.3 provides:

Corollary 4.2. *For any fixed $n \geq 3, s \geq n - 2$ and rational $t \geq 0$, n−s−t−Split−Bound($\mathcal{X}$) can be solved in polynomial time.*

Finally, for any fixed integer $n \geq 2$ and $s \geq n - 2$, we can solve the n−s−Split−MinMax ($\mathcal{X}$) optimization problem by using binary search on the parameter t of the n−s−t−Split −Bound($\mathcal{X}$) problem. The details are given in Appendix K. As explained there, the binary search procedure needs to solve at most $\log_2(nS)$ instances of n−s−t−Split−Bound($\mathcal{X}$).

Corollary 4.3 (Theorem 1.1)**.** *For any fixed integers $n \geq 3$ and $s \geq n - 2$, the problem n−s−Split−MinMax($\mathcal{X}$) can be solved in polynomial time in the size of the binary representation of its input.*

5 Conclusion and Future Directions

We presented two variants of the multiway number partitioning problem. In the language of machine scheduling, n−s−Split−MinMax corresponds to finding a schedule that minimizes the makespan on n identical machines when s jobs can be split between the machines; n−d−Interval corresponds to finding a schedule in which the makespan is in a given interval. It may be interesting to study these two variants for more general scheduling problems, such as *uniform machines* and *unrelated machines.*

In the language of fair item allocation, we have solved the problem of finding a fair allocation among n agents with identical valuations, when the ownership on some s items may be shared among agents. With identical valuations, “fair” simply means that each agent receives the same sum of values. When agents may have different valuations, there are various generalizations to this fairness notion, such as *proportionality*, *envy-freeness* or *equitability.* A future research direction is to develop algorithms for finding such allocations with a bounded number of shared items.

Our analysis shows the similarities and differences between these two variants and the more common notion of FPTAS. One may view our results as introducing two alternative kinds of approximation.

The first keeps the objective value optimal and allows a bounded number of items to become fractional, instead of keeping the items discrete and allowing a bounded deviation in the objective value. More generally, consider any optimization problem that can be solved by a mixed-integer program, where some of the problem variables are rationals and some are integers. Instead of deciding in advance (in the problem definition) which of the variables must be integers, one can consider a variant in which only the *number* of the integer variables is fixed in advance, whereas the solver can decide *which* variables are integers after receiving the input.

The second approximates a decision problem by returning “yes” iff there exists a solution between PER and $PER(1 + \epsilon)$, where PER represents the value of a perfect solution. For the n−Way Number Partitioning problem, a perfect solution is easy to define: it is a perfect partition. A more general definition of PER could be the solution to the fractional relaxation of an integer linear program representing the problem. As we have shown, NP-hard decision problems may become tractable when ϵ is sufficiently large.

References

[1] Xiaohui Bei, Zihao Li, Jinyan Liu, Shengxin Liu, and Xinhang Lu. Fair division of mixed divisible and indivisible goods. *Artif. Intell.*, 293:103436, 2021.

[2] Luca Bertazzi, Bruce L. Golden, and Xingyin Wang. The bin packing problem with item fragmentation: A worst-case analysis. *Discret. Appl. Math.*, 261:63–77, 2019. doi: 10.1016/j.dam.2018.08.023. URL `https://doi.org/10.1016/j.dam.2018.08.023`.

[3] Steven J. Brams and Alan D. Taylor. *Fair Division: From Cake Cutting to Dispute Resolution.* Cambridge University Press, Cambridge UK, February 1996. ISBN 0521556449.

[4] Steven J. Brams and Alan D. Taylor. *The win-win solution - guaranteeing fair shares to everybody.* W. W. Norton & Company, 2000. ISBN 978-0-393-32081-7.

[5] Steven J. Brams and Jeffrey M. Togman. Camp David: Was The Agreement Fair? *Conflict Management and Peace Science*, 15(1):99–112, February 1996. ISSN 1549-9219.

[6] Alberto Caprara, Hans Kellerer, and Ulrich Pferschy. The multiple subset sum problem. *SIAM Journal on Optimization*, 11(2):308–319, 2000.

[7] Terry Daniel and James Parco. Fair, Efficient and Envy-Free Bargaining: An Experimental Test of the Brams-Taylor Adjusted Winner Mechanism. *Group Decision and Negotiation*, 14(3):241–264, May 2005. ISSN 0926-2644.

[8] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979. ISBN 0-7167-1044-7.

[9] Paul W. Goldberg, Alexandros Hollender, Ayumi Igarashi, Pasin Manurangsi, and Warut Suksompong. Consensus halving for sets of items. In Xujin Chen, Nikolai Gravin, Martin Hoefer, and Ruta Mehta, editors, *Web and Internet Economics - 16th International Conference, WINE 2020, Beijing, China, December 7-11, 2020, Proceedings*, volume 12495 of *Lecture Notes in Computer Science*, pages 384–397. Springer, 2020.

[10] Katrin Heßler, Stefan Irnich, Tobias Kreiter, and Ulrich Pferschy. Bin packing with lexicographic objectives for loading weight- and volume-constrained trucks in a direct-shipping system. *OR Spectr.*, 44(2):1–43, 2022. doi: 10.1007/s00291-021-00628-x. URL `https://doi.org/10.1007/s00291-021-00628-x`.

[11] Richard E. Korf. A complete anytime algorithm for number partitioning. *Artif. Intell.*, 106(2):181–203, 1998. doi: 10.1016/S0004-3702(98)00086-1. URL `https://doi.org/10.1016/S0004-3702(98)00086-1`.

[12] Zhaohui Liu and T.C.Edwin Cheng. Scheduling with job release dates, delivery times and preemption penalties. *Information Processing Letters*, 82(2):107–111, 2002. ISSN 0020-0190.

[13] Enrico Malaguti, Michele Monaci, Paolo Paronuzzi, and Ulrich Pferschy. Integer optimization with penalized fractional values: The knapsack case. *Eur. J. Oper. Res.*, 273 (3):874–888, 2019. doi: 10.1016/j.ejor.2018.09.020. URL `https://doi.org/10.1016/j.ejor.2018.09.020`.

[14] Tansa G. Massoud. Fair Division, Adjusted Winner Procedure (AW), and the Israeli-Palestinian Conflict. *Journal of Conflict Resolution*, 44(3):333–358, June 2000. ISSN 1552-8766.

[15] Nir Menakerman and Raphael Rom. Bin packing with item fragmentation. In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Roberto Tamassia, editors, *Algorithms and Data Structures, 7th International Workshop, WADS 2001, Providence, RI, USA, August 8-10, 2001, Proceedings*, volume 2125 of *Lecture Notes in Computer Science*, pages 313–324. Springer, 2001. doi: 10.1007/3-540-44634-6_29. URL `https://doi.org/10.1007/3-540-44634-6_29`.

[16] Elisha A. Pazner and David Schmeidler. Egalitarian Equivalent Allocations: A New Concept of Economic Equity. *Quarterly Journal of Economics*, 92(4):671–687, November 1978. ISSN 1531-4650.

[17] Fedor Sandomirskiy and Erel Segal-Halevi. Efficient fair division with minimal sharing, 2020. arXiv preprint 1908.01669.

[18] Gerald Schneider and Ulrike S. Krämer. The Limitations of Fair Division. *Journal of Conflict Resolution*, 48(4):506–524, August 2004. ISSN 1552-8766.

[19] Ethan L. Schreiber, Richard E. Korf, and Michael D. Moffitt. Optimal multi-way number partitioning. *J. ACM*, 65(4):24:1–24:61, 2018. doi: 10.1145/3184400. URL `https://doi.org/10.1145/3184400`.

[20] Erel Segal-Halevi. Fair division with bounded sharing. *CoRR*, abs/1912.00459, 2019. URL `http://arxiv.org/abs/1912.00459`.

[21] Alan J. Soper and Vitaly A. Strusevich. Schedules with a single preemption on uniform parallel machines. *Discrete Applied Mathematics*, 261:332–343, 2019. ISSN 0166-218X. GO X Meeting, Rigi Kaltbad (CH), July 10–14, 2016.

[22] Stephen J Wilson. Fair division using linear programming. *preprint, Departement of Mathematics, Iowa State University*, 1998.

[23] Gerhard J. Woeginger. When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? *INFORMS J. Comput.*, 12(1):57–74, 2000.

Samuel Bismuth
Department of Computer Science, Ariel University
Ariel 40700, Israel
Email: `samuelbismuth101@gmail.com`

Vladislav Makarov
Department of Mathematics and Computer Science, St. Petersburg State University
St. Petersburg 199178
Email: `vm450@yandex.ru`

Erel Segal-Halevi
Department of Computer Science, Ariel University
Ariel 40700, Israel
Email: `erelsgl@gmail.com`

Dana Shapira
Department of Computer Science, Ariel University
Ariel 40700, Israel
Email: `shapird@g.ariel.ac.il`

APPENDIX

A Hardness for $n \geq 3$ bins and $d < n-2$

The following theorem complements the Section 3.2

Theorem A.1 (Theorem 1.5)**.** *Given a fixed integer $n \geq 3$ and a positive rational number $d < n-2$, the problem n–d–Interval($\mathcal{X}$) is NP-complete.*

Proof. Given an n-way partition of m items, summing the sizes of all elements in each bin allows us to check whether the partition is d-possible in linear time. So, the problem is in NP.

To prove that n–d–Interval is NP-Hard, we reduce from the Equal-Cardinality Partition problem, proved to be NP-hard in [8]: given a list with an even number of integers, decide if they can be partitioned into two subsets with the same sum and the same cardinality.

Given an instance $\mathcal{X}_1$ of Equal-Cardinality Partition, denote the number of items in $\mathcal{X}_1$ by $2m'$. Define M to be the sum of numbers in $\mathcal{X}_1$ divided by $2n(1-\frac{d}{n-2})$, so that the sum of items in $\mathcal{X}_1$ is $2n(1-\frac{d}{n-2})M$ (where n and d are the parameters in the theorem statement). We can assume w.l.o.g. that all items in $\mathcal{X}_1$ are at most $n(1-\frac{d}{n-2})M$, since if some item is larger than half of the sum, the answer is necessarily "no".

Construct an instance $\mathcal{X}_2$ of the Equal-Cardinality Partition problem by replacing each item x in $\mathcal{X}_1$ by $nM - x$. So $\mathcal{X}_2$ contains $2m'$ items between $n(\frac{d}{n-2})M$ and nM. Their sum, which we denote by $2S'$, satisfies

$$2S' = 2m' \cdot nM - 2n\left(1 - \frac{d}{n-2}\right)M = 2n\left(m' - 1 + \frac{d}{n-2}\right)M.$$

Clearly, $\mathcal{X}_1$ has an equal-sum equal-cardinality partition (with bin sums $n\left(1-\frac{d}{n-2}\right)M$) if and only if $\mathcal{X}_2$ has an equal-sum equal-cardinality partition (with bin sums $S' = n\left(m'-1+\frac{d}{n-2}\right)M$).

Construct an instance $(\mathcal{X}_3, d)$ of n–d–Interval by adding $(n-2)(m'-1)$ items of size nM. Note that nM is indeed the largest item size in $\mathcal{X}_3$. Denote the sum of item sizes in $\mathcal{X}_3$ by nS. Then

$$\begin{aligned} nS &= 2S' + (n-2)(m'-1) \cdot nM \\ &= n\left(2(m'-1) + \frac{2d}{n-2} + (n-2)(m'-1)\right) \cdot M \\ &= n\left(n(m'-1) + \frac{2d}{n-2}\right)M; \end{aligned}$$

$$S + dM = \left(n(m'-1) + \frac{2d}{n-2} + d\right)M = \left(n(m'-1) + \frac{nd}{n-2}\right)M = S',$$

so a partition of $\mathcal{X}_3$ is d-possible if and only if the sum of each of the n bins in the partition is at most $S + dM = S'$.

We now prove that if $\mathcal{X}_2$ has an equal-sum equal-cardinality partition, then the corresponding instance $(\mathcal{X}_3, d)$ has a d-possible partition, and vice versa.

If $\mathcal{X}_2$ has an equal-sum partition, then the items of $\mathcal{X}_2$ can be partitioned into two bins of sum S', and the additional $(n-2)(m'-1)$ items can be divided into $n-2$ bins of $m'-1$ items each. Note that the sum of these items is

$$(m'-1) \cdot nM = n(m'-1)M = S - \frac{2}{n-2}dM < S + dM = S', \tag{1}$$

so the resulting partition is a d-possible partition of $\mathcal{X}_3$. Conversely, suppose $\mathcal{X}_3$ has a d-possible partition. Let us analyze its structure.

- Since the partition is d-possible, the sum of every two bins is at most $2(S + dM)$.
- So the sum of every $n - 2$ bins is at least $nS - 2(S + dM) = (n - 2)S - 2dM$.
- Since the largest $(n - 2)(m' - 1)$ items in $\mathcal{X}_3$ sum up to exactly $(n - 2)S - 2dM$ by (2), every $n - 2$ bins must contain at least $(n - 2)(m' - 1)$ items.
- Since $\mathcal{X}_3$ has $(n - 2)(m' - 1) + 2m'$ items overall, $n - 2$ bins must contain exactly $(n - 2)(m' - 1)$ items, such that each item size must be nM, and their sum must be $(n - 2)S - 2dM$.
- The other two bins contain together $2m'$ items with sum $2(S + dM)$, so each of these bins must have a sum of exactly $S + dM$. Since $(m' - 1) \cdot nM < S + dM$ by (2), each of these two bins must contain exactly m' items.

These latter two bins are an equal-sum equal-cardinality partition for $\mathcal{X}_2$. This construction is done in polynomial time, completing the reduction. $\square$

B Strong hardness for non-fixed n

In this section we prove that when n is part of the input (and not a fixed parameter), the problems $d-\mathsf{Interval}(\mathcal{X}, n)$ and $s-\mathsf{Split-Perfect}(\mathcal{X}, n)$ are strongly NP-hard.

First, we prove that the problem $d-\mathsf{Interval}(\mathcal{X}, n)$ is strongly NP-hard. Then, we show that $s-\mathsf{Split-Perfect}(\mathcal{X}, n)$ is strongly NP-hard by reduction from $d-\mathsf{Interval}(\mathcal{X}, n)$.

Theorem B.1. *For every fixed rational number $d \geq 0$, The problem $d-\mathsf{Interval}(\mathcal{X}, n)$ is strongly NP-hard.*

Proof. We apply a reduction from the $3-\mathsf{Partition}$ problem: given a list $\mathcal{X}_1$ of $3m'$ positive integers with sum equal to $m'S'$, decide if they can be partitioned into m' triplets such that the sum of each triplet is S'. $3-\mathsf{Partition}$ is proved to be strongly NP-hard in [8].

Given an instance $\mathcal{X}_1$ of $3-\mathsf{Partition}$ with $3m'$ integers, define $n := \lceil 2(m' + d) \rceil$, where d is the parameter of the problem $d-\mathsf{Interval}$. Define M to be the sum of numbers in $\mathcal{X}_1$ divided by $m'n(1 - \frac{d}{n-m'})$, so that the sum of items in $\mathcal{X}_1$ is $m'n(1 - \frac{d}{n-m'})M$. Since $n = \lceil 2(m' + d) \rceil$, it follows that $1 - \frac{d}{n-m'} > 0$, so M is positive. The $3-\mathsf{Partition}$ problem decides if the items can be partitioned into m' triplets such that the sum of each triplet is $n(1 - \frac{d}{n-m'})M$. We can assume w.l.o.g. that all items in $\mathcal{X}_1$ are at most $n(1 - \frac{d}{n-m'})M$, since if some item is larger than $n(1 - \frac{d}{n-m'})M$, the answer is necessarily "no".

Construct an instance $\mathcal{X}_2$ of the $3-\mathsf{Partition}$ problem by replacing each item x in $\mathcal{X}_1$ by $nM - x$. So $\mathcal{X}_2$ contains $3m'$ items between $n(\frac{d}{n-m'})M$ and nM. Their sum, which we denote by $m'S'$, satisfies

$$m'S' = 3m' \cdot nM - m'n\left(1 - \frac{d}{n - m'}\right)M = m'n\left(2 + \frac{d}{n - m'}\right)M.$$

Clearly, $\mathcal{X}_1$ has a three-partition (with sum $n\left(1 - \frac{d}{n-m'}\right)M$) if and only if $\mathcal{X}_2$ has a three-partition (with sum $S' = n\left(2 + \frac{d}{n-m'}\right)M$). The construction is done in time polynomial in the size of $\mathcal{X}_1$.

Construct an instance $(\mathcal{X}_3, n)$ of $d-$Interval by adding $2(n-m')$ items of size nM. Note that nM is indeed the largest item size in $\mathcal{X}_3$. Denote the sum of item sizes in $\mathcal{X}_3$ by nS. So

$$\begin{aligned} nS &= m'S' + 2(n-m') \cdot nM = n\left(2m' + \frac{m'd}{n-m'} + 2(n-m')\right) \cdot M \\ &= n\left(2n + \frac{m'd}{n-m'}\right) M; \\ S + dM &= \left(2n + \frac{m'd}{n-m'} + d\right) M = \left(2n + \frac{nd}{n-m'}\right) M = S', \end{aligned}$$

so a partition of $\mathcal{X}_3$ is d-possible if and only if the sum of each of the n bins in the partition is at most $S + dM = S'$.

We now prove that if $\mathcal{X}_2$ has a three-partition then the corresponding instance $(\mathcal{X}_3, n)$ has a d-possible partition, and vice versa.

If $\mathcal{X}_2$ has a three-partition, then the items of $\mathcal{X}_2$ can be partitioned into m' bins of sum S', and the additional $2(n-m')$ items can be divided into $n-m'$ bins of 2 items each. Note that the sum of 2 additional items is

$$\begin{aligned} 2 \cdot nM &= \left(2n + \frac{nd}{n-m'}\right) M - \frac{nd}{n-m'} M = S + dM - \frac{nd}{n-m'} M \\ &= S + (d - \frac{nd}{n-m'})M = S - \frac{m'}{n-m'} dM < S + dM = S', \end{aligned} \tag{2}$$

where the last inequality follows from the fact that $0 < \frac{m'}{n-m'}$, so the resulting partition is a d-possible partition of $\mathcal{X}_3$.

Conversely, suppose $\mathcal{X}_3$ has a d-possible partition. Let us analyze its structure.

– Since the partition is d-possible, the sum of every m' bins is at most $m'(S + dM)$.

– So the sum of every $n - m'$ bins is at least $nS - m'(S + dM) = (n - m')S - m'dM$.

– The largest $2(n-m')$ items in $\mathcal{X}_3$ sum up to exactly $2(n-m') \cdot nM = (n-m')S - m'dM$ by (2).

– Hence, every $n - m'$ bins must contain *at least* $2(n - m')$ items.

– Let A be the set of $n - m'$ bins with the fewest items. We claim that A must contain *exactly* $2(n - m')$ items. Suppose by contradiction that A contained at least $2(n - m') + 1$ items. By the pigeonhole principle, there was a bin in A with at least 3 items. By minimality of A, the m' bins not in A also contained at least 3 items each. The total number of items were $2(n - m') + 1 + 3m'$. This is a contradiction, since the total number of items in $\mathcal{X}_3$ is only $2(n - m') + 3m'$.

– The sum of items in A must still be at least $(n - m')S - m'dM = 2(n - m') \cdot nM$, so each item in A must have the maximum size of nM.

– The other m' bins contain together $3m'$ items with sum $m'(S + dM)$, so each of these bins must have a sum of exactly $S + dM$. Since $2 \cdot nM < S + dM$ by (2), each of these m' bins must contain exactly 3 items.

These latter m' bins are a three-partition of $\mathcal{X}_2$. This construction is done in time polynomial in the size of $\mathcal{X}_2$, since we added $2(n-m')$ items to the initial $3m'$ items, so the new number of items is $m' + 2n < m' + 4(m' + d) + 1 = 5m' + 4d + 1$. For every constant d, this number is linear in the size of $\mathcal{X}_2$. $\square$

Theorem B.2. *For any fixed integer $s = d \geq 0$, there is a polynomial-time reduction from d–$\mathsf{Interval}(\mathcal{X}, n)$ to s–Split–$\mathsf{Perfect}(\mathcal{X}, n)$.*

Proof. Given an instance $(\mathcal{X}, n)$ of d–$\mathsf{Interval}(\mathcal{X}, n)$, construct an instance $(\mathcal{X}', n')$ of s–Split –$\mathsf{Perfect}(\mathcal{X}', n')$ by setting $n' = n$ and adding $d = s$ items of size nM, where nM is the size of the biggest item in $\mathcal{X}$.

First, assume that $(\mathcal{X}, n)$ has a d-possible partition. Then there are n bins with a sum at most $S + dM$, where S is the sum of items in $\mathcal{X}$ divided by n. Take the d added items of size nM and add them to the bins, possibly splitting some items between bins, such that the sum of each bin becomes exactly $S + dM$. This is possible because the sum of the items in $(\mathcal{X}', n')$ is $nS + dnM = n(S + dM)$. The result is a perfect partition of $(\mathcal{X}', n')$ with at most $d = s$ split items.

Second, assume that $(\mathcal{X}', n')$ has a perfect partition with at most s split items. Then there are n' bins with sum exactly $S + dM$. By Lemma 4.1, we can assume that the split items are the largest ones, which are the d added items of size nM. Remove these items to get a partition of $(\mathcal{X}, n)$. The sum in each bin is now at most $S + dM$, so the partition is d-possible.

This construction is done in polynomial time, which completes the proof. ☐

Combining Theorem B.1 and Theorem B.2 gives:

Corollary B.1. *For every fixed integer $s \geq 0$, the problem s–Split–$\mathsf{Perfect}(\mathcal{X}, n)$ is strongly NP-hard.*

C Hardness with Splitting count

In this section we analyze the complexity of partitioning when the parameter s denotes the number of *splittings*, rather than the number of *split items*.

Theorem C.1. *For any fixed integer $n \geq 2$ and fixed $s \in \mathbb{N}$ such that $s \leq n-2$, the problem n–s–Times–Split–$\mathsf{Perfect}(\mathcal{X})$ is NP-complete.*

Proof. Given a partition with n bins, m items and s splittings, summing the size of each element (or fraction of element) in each bin allows us to check whether or not the partition is perfect in linear time. So, the problem is in NP.

To prove that n–s–Times–Split–$\mathsf{Perfect}$ is NP-Hard, we apply a reduction from the $\mathsf{Subset\ Sum}$ problem. We are given an instance $\mathcal{X}_1$ of $\mathsf{Subset\ Sum}$ with m items summing up to S and target sum $T < S$. We build an instance $\mathcal{X}_2$ of n–s–Times–Split–$\mathsf{Perfect}(\mathcal{X}_2)$, by adding two items, x_1, x_2, such that $x_1 = S + T$ and $x_2 = 2S(s+1) - T$ and $n - 2 - s$ auxiliary items of size $2S$. Notice that the sum of the items in $\mathcal{X}_2$ equals

$$\begin{aligned}
&S + (S + T) + 2S(s + 1) - T + 2S(n - 2 - s) \\
&= 2S + 2S(s + 1) + 2S(n - 2 - s) \\
&= 2S \cdot (1 + s + 1 + n - 2 - s) \\
&= 2Sn.
\end{aligned}$$

The goal is to partition the items into n bins with a sum of $2S$ per bin, and at most s splittings.

First, assume that there is a subset of items W_1 in $\mathcal{X}_1$ with sum equal to T. Define a set, W_2, of items that contains all items in $\mathcal{X}_1$ that are not in W_1, plus x_1. The sum of W_2 is $(S - T) + x_1 = S + T + S - T = 2S$. Assign the items of W_2 to the first bin. Assign each auxiliary item to a different bin. There are $n - (n - 2 - s + 1) = s + 1$ bins left. The sum of the remaining items is $2S(s+1)$. Using the "line-cutting" algorithm described in the introduction, these items can be partitioned into $s+1$ bins of equal sum $2S$, with at most s splittings. All in all, there are n bins with a sum of $2S$ per bin, and the total number of splittings is at most s.

Second, assume that there exists an equal partition for n bins with s splittings. Since $x_2 = 2S(s+1) - T = 2S \cdot s + (2S - T) > 2S \cdot s$, this item must be split between $s+1$ bins, which makes the total number of splittings at least s. Also, the auxiliary items must be assigned without splittings into $n-2-s$ different bins. There is $n-s-1-n+2+s = 1$ bin remaining, say bin i, containing only whole items, not containing any part of x_2, and not containing any auxiliary item. Bin i must contain x_1, otherwise its sum is at most S (sum of items in $\mathcal{X}_1$). Let W_1 be the items of $\mathcal{X}_1$ that are not in bin i. The sum of W_1 is $S - (2S - x_1) = x_1 - S = T$, so it is a solution to $\mathcal{X}_1$. $\square$

Note that a weaker result was presented in [20], that n–s–Times–Split–Perfect($\mathcal{X}$) is NP-complete for $s < n-2$.

D Practical examples

As a first example, consider a food factory with $n = 3$ identical chopping machines, who has to cut $m = 4$ vegetables with processing times $\mathcal{X} = (10, 7, 5, 5)$ minutes.

Each job is divisible — as one vegetable may be cut in different machines, but splitting a job is inconvenient — since it requires washing more dishes. Without splitting, the minimum total processing time is 10 minutes: $(10), (7), (5, 5)$. By splitting the vegetable with processing time 10 into three different machines, the processing time is 9 minutes: $(7, 2), (5, 4), (5, 4)$.

As a second example (using the same m, n and $\mathcal{X}$), a video streaming platform earns money by showing advertisements to users. The platform has an agreement with m different companies to display $\mathcal{X}$ seconds of advertisements to n users. One or several of them can be split by enabling the user to skip it after a given number of seconds. Nevertheless, if an advertisement is split, the platform earns less. Similarly to the case with the chopping machine, the option to split a single advertisement can make the division more balanced among the users.

E FPTAS-s for various partitioning problems

Our algorithms use FPTAS-s for several variants of the number partitioning problem. These FPTAS-s are developed using a general technique described by Woeginger [23]. In this section we briefly describe this technique.

The first step is to develop a dynamic programming (DP) algorithm that solves the problem exactly. The algorithm should have a specific format called *simple DP*, defined by the following parameters:

- The size of the input vectors, $\alpha \in \mathbb{N}$ (in our settings, usually $\alpha = 1$ since each input is a single integer);
- The size of the *state vectors*, $\beta \in \mathbb{N}$ (in our settings, each state represents a partition of a subset of the inputs);
- A set of initial states V_0 (in our settings, V_0 usually contains a single state — the zero vector — representing the empty partition);

– A set of *transition functions* F; each function in F accepts a state and an input, and returns a new state (in our settings, each function in F corresponds to putting the new input in one of the bins);

– A set of *filter functions* H; each function $h_j \in H$ corresponds to a function $f_j \in F$. It accepts a state and an input, and returns a positive value if the new state returned by f_j is infeasible (infeasible states are kept out of the state space).

– An *objective function* G, that maps a state to a numeric value.

The DP algorithm processes the inputs one by one. For each input k, it applies every transition function in F to every state in V_{k-1}, to produce the new state-set V_k. Finally, it picks the state in V_m that minimizes the objective function. Formally:

1. Let V_0 be the set of initial states.
2. For $k := 1, \ldots, m$:
$$V_k := \{f_j(s,x) | f_j \in F,\ s \in V_{k-1},\ h_j(s,x) \leq 0\}$$
3. Return
$$\min\{G(s) | s \in V_m\}$$

Every DP in this format can be converted to an FPTAS if it satisfies a condition called *critical-coordinate-benevolence* (CC-benevolence). To prove that a dynamic program is CC-benevolent, we need to define a *degree vector* D of size β, which determines how much each state is allowed to deviate from the optimal state. The functions in F, G, H and the vector D should satisfy several conditions listed in Lemma 6.1 of [23]. These conditions use the term $[D, \Delta]$-*close*, defined as follows.

For a real number $\delta > 1$ and two vectors $V = [v_1, \ldots, v_\beta]$ and $V' = [v'_1, \ldots, v'_\beta]$, we say that V is $[D, \Delta]-$close to V' if

$$\Delta^{-d_\ell} \cdot v_\ell \ \leq\ v'_\ell \ \leq\ \Delta^{d_\ell} \cdot v_\ell, \text{ for all } \ell = 1, \ldots, \beta,$$

that is, each coordinate in V' deviates from the corresponding coordinate in V by a multiplicative factor determined by Δ and by the degree vector D. The Δ is a factor determined by the required approximation accuracy ϵ. Note that if some coordinate ℓ in D is 0, then the definition of $[D, \Delta]$-close requires that the coordinate ℓ in V' is equal to coordinate ℓ in V (no deviation is allowed).

We now use Lemma 6.1 of [23] to prove that the problems $n-t-$Way-MinMax$-$CC and $n-t-$Interval$-$Eq are CC-benevolent, and therefore there is an FPTAS for solving these problems.

E.1 $n-t-$Way-MinMax$-$CC

Recall that the objective of $n-t-$Way-MinMax$-$CC$(\mathcal{X})$ is to find an n-way partition of $\mathcal{X}$ minimizing the largest bin sum, subject the the constraint that the sum of bin #1 is at most $(1+t) \cdot S$, where $t \in \mathbb{Q}$ is given in the input and $S = (\sum_{x \in \mathcal{X}} x)/n$ is the average bin sum.

Proof. **The dynamic program**. We define a simple DP with $\alpha = 1$ (the size of the input vectors) and $\beta = n$ (the size of the state vectors). For $k = 1, \ldots, m$ define the input vector $U_k = [x_k]$ where x_k is the size of item k. A state $V = [b_1, b_2, \ldots, b_n]$ in V_k encodes a partial allocation for the first k items, where b_i is the sum of items in bin $i \in \{1, 2 \ldots, n\}$ in the partial allocation. The set F contains n transition functions $f_1, \ldots, f_n$:

$$\begin{aligned}
&f_1(x_k, b_1, b_2, \ldots, b_n) = [b_1 + x_k, b_2, \ldots, b_n] \\
&f_2(x_k, b_1, b_2, \ldots, b_n) = [b_1, b_2 + x_k, \ldots, b_n] \\
&\ldots \\
&f_n(x_k, b_1, b_2, \ldots, b_n) = [b_1, b_2, \ldots, b_n + x_k]
\end{aligned}$$

Intuitively, the function f_i corresponds to putting item k in bin i. The set H contains a function $h_1(x_k, b_1, b_2, \ldots, b_n) = b_1 + x_k - (1+t) \cdot S$, and functions $h_i(x_k, b_1, b_2, \ldots, b_n) \equiv 0$ for $i \in \{2, \ldots, n\}$. These functions represent the fact that the sum of the first bin must always be at most $(1+t) \cdot S$ (if it becomes larger than $(1+t) \cdot S$, then h_1 will return a positive value and the new state will be filtered out). There are no constraints on the sums of bins $2 \ldots, n$.

The initial state space V_0 contains a single vector $\{[0, 0, \ldots, 0]\}$. The minimization objective is

$$G(b_1, b_2, \ldots, b_n) = \max\{b_1, b_2, \ldots, b_n\}.$$

Benevolence. We show that our problem is *CC-benevolent*, as defined at [23][Section 6]. We use the degree vector $D = [1, 1, \ldots, 1]$, and define b_1 as the critical coordinate.

All the transition functions are polynomials of degree 1. The value of the function $f_1(U, V)$ only depends on x_k which is on U, and b_1 which is our critical-coordinate. So, C.1(i) on the function set F is fulfilled.

The functions $h_1, h_2, \ldots, h_n$ are polynomials; the monomials do not depend on $b_2, \ldots, b_n$, and the monomial that depends on b_1 has a positive coefficient. So, C.2(i) on the function set H is fulfilled.

If a state $[b_1, b_2, \ldots, b_n]$ is $[D, \Delta]$-close to another state $[b'_1, b'_2, \ldots, b'_n]$, then by $[D, \Delta]$-close definition, we have $v'_1 \leq \Delta \cdot v_1$, $v'_2 \leq \Delta \cdot v_2$, $\ldots$, $v'_n \leq \Delta \cdot v_n$, so

$$\max\{b'_1, b'_2, \ldots, b'_n\} \leq \Delta \cdot \max\{b_1, b_2, \ldots, b_n\},$$

and $G(V') \leq \Delta \cdot G(V)$ Therefore, C.3(i) on the function G is fulfilled (with degree $g = 1$).

The definition in [23] also allows a *domination relation*, but we do not need it in our case (formally, our domination relation $\preceq_{dom}$ is the trivial relation). So, the statement conditions C.1(ii), C.2(ii), C.3(ii) are fulfilled.

Condition C.4 (i) holds since all functions in F can be evaluated in polynomial time. C.4 (ii) holds since the cardinality of F is a constant. C.4 (iii) holds since the cardinality of V_0 is a constant. C.4(iv) is satisfied since the value of the coordinates is upper bounded by the sum of the items nS, so their logarithm is bounded by the size of the input. Hence, our problem is CC-benevolent. By the main theorem of [23], it has an FPTAS. □

E.2 2–t–Way-MinMax–Eq–CC

Recall that the objective is to find a 2-way partition of $\mathcal{X}$ minimizing the largest bin sum, subject the the constraint that the sum of bin #1 is at most $(1+t) \cdot S$, and additionally, the number of items in each bin is the same.

Proof. **The dynamic program**. We define a simple DP with $\alpha = 1$ (the size of the input vectors) and $\beta = 4$ (the size of the state vectors). For $k = 1, \ldots, m$ define the input vector $U_k = [x_k]$, where x_k is the size of item k. A state $V = [b_1, b_2, l_1, l_2]$ in V_k encodes a partial allocation for the first k items, where b_i is the sum of items in bin $i \in \{1, 2\}$ in the partial allocation and l_i is the number of items in bin i. The set F contains two transition functions F_1 and F_2:

$$f_1(x_k, b_1, b_2, l_1, l_2) = [b_1 + x_k, b_2, l_1 + 1, l_2]$$
$$f_2(x_k, b_1, b_2, l_1, l_2) = [b_1, b_2 + x_k, l_1, l_2 + 1]$$

Intuitively, the function f_i corresponds to putting item k in bin i. Let H be a set of two functions h_1 and h_2. The function $h_1(x_k, b_1, b_2, l_1, l_2) = b_1 + x_k - (1+t)S$ corresponds to an upper bound of $(1+t)S$ on the sum of the first bin, and a function $h_2(x_k, b_1, b_2, l_1, l_2) \equiv 0$ that corresponds to having no upper bound on the sum of the second bin. Finally, set the minimization objective to

$$G(b_1, b_2, l_1, l_2) = \begin{cases} \max\{b_1, b_2\}, & \text{if } l_1 = l_2 \\ \infty, & \text{otherwise} \end{cases}$$

The initial state space V_0 is set to $\{[0, 0, 0, 0]\}$.

Benevolence. We show that our problem is *CC-benevolent*, as defined at [23][Section 6].

We define the degree vector as $D = [1, 1, 0, 0]$. Note that the third and fourth coordinates correspond to the number of items in each bin, for which we need an exact number and not an approximation.

Lemma 4.1(i) is satisfied since if a state $[b_1, b_2, l_1, l_2]$ is $[D, \Delta]$-close to another state $[b'_1, b'_2, l'_1, l'_2]$, by definition of $[D, \Delta]$-close [Section 2](2.1), we must have $l_1 = l'_1$ and $l_2 = l'_2$ because the degree of coordinates 3 and 4 is 0. At coordinates 1 and 2, both transition functions are polynomials of degree 1. Furthermore, the value of the function $f_1(U, V)$ only depends on x_k which is on U and b_1 which is our critical-coordinate. So, C.1(i) on the function set F is fulfilled.

The functions h_1 and h_2 are polynomials; the monomials do not depend on b_2, and the monomial that depends on b_1 has a positive coordinate. So, C.2(i) on the function set H is fulfilled.

If a state $[b_1, b_2, l_1, l_2]$ is $[D, \Delta]$-close to another state $[b'_1, b'_2, l'_1, l'_2]$ (where Δ is a factor determined by the required approximation accuracy ϵ), by $[D, \Delta]$-close definition, we have $l_1 = l'_1$ and $l_2 = l'_2$ because the degree of coordinates 3 and 4 is 0, also $b'_1 \le \Delta b_1$ and $b'_2 \le \Delta b_2$, so $\max\{b'_1, b'_2\} \le \Delta \max\{b_1, b_2\}$, so $G(V') \le \Delta^g G(V)$. Therefore, C.3(i) on the function G is fulfilled (with degree $g = 1$).

Again we do not need a domination relation, so we use the trivial relation $\preceq_{dom}$. Then the statement conditions C.1(ii), C.2(ii), C.3(ii) are fulfilled.

Condition C.4 (i) holds since all functions in F can be evaluated in polynomial time. C.4 (ii) holds since the cardinality of F is a constant. C.4 (iii) holds since the cardinality of V_0 is a constant. C.4(iv) is satisfied for coordinates 1, 2 since their value is upper bounded by the sum of the items nS, so their logarithm is bounded by the size of the input. For coordinates 3, 4 (whose degree is 0), the condition is satisfied since their value is upper bounded by the number of items m. Hence, our problem is CC-benevolent. By the main theorem of [23], it has an FPTAS. $\square$

E.3 Running time

We compute in this section the running time of Algorithm 1 and Algorithm 2. The running time of each FPTAS is linear in the number of states in the dynamic program. The number of states in the dynamic program is bounded by L^n, where L is the number of possible values in every element of the vector. In [23][Section 3], Woeginger proves that $L \le \lceil(1 +$

$\frac{2gm}{\epsilon})\pi_1(m, \log_2 \bar{x})\rceil$, where $m \in \mathbb{N}$ is the number of items, $\bar{x}$ is the sum of all item sizes (which in our case is just nS), and π_1 is a function describing the binary length of the values in the state vectors (in our case it is a linear function).

In each FPTAS settings, $g = 1$, $\bar{x} = \sum_{k=1}^{m} \sum_{i=1}^{\alpha} x_{i,k}$, where $\alpha = 1$, then, $\bar{x}$ is the sum of all the items. Because every value in the state-vectors is at most the sum of all values, we have the linear function $\pi_1(m, \log_2 \bar{x}) = \log_2 \bar{x}$.

E.3.1 Algorithm 1 running time

Algorithm 1 runs an FPTAS for $2-t-$Way-MinMax$-$CC. In $2-t-$Way-MinMax$-$CC, $\epsilon = t/2$ and the sum of the items $\bar{x}$, is equal to $2S$. Therefore, we have the following: $L \leq \lceil(1 + \frac{2m}{t/2})\log_2(2S)\rceil = \lceil(1 + \frac{4m}{t})\log_2(2S)\rceil \in O(\frac{m}{t}\log S)$.

Therefore, Algorithm 1 running time is $O(\frac{m}{t}\log S)$.

E.3.2 Algorithm 2 running time

Algorithm 2 runs two FPTAS, one for $n-t-$Way-MinMax$-$CC and one for $2-t-$Way-MinMax $-$Eq$-$CC. In $n-t-$Way-MinMax$-$CC and for $2-t-$Way-MinMax$-$Eq$-$CC, $\epsilon = t/4m^2$. In $n-t-$Way-MinMax$-$CC the sum of the items $\bar{x}$, is equal to nS while in $2-t-$Way-MinMax $-$Eq$-$CC the sum of the items $\bar{x}$, is equal to $2S$.

Therefore, we have the following:

- for $n-t-$Way-MinMax$-$CC: $L \leq \lceil(1 + \frac{2m}{t/4m^2})\log_2(nS)\rceil = \lceil(1 + \frac{8m^3}{t})\log_2(nS)\rceil \in O(\frac{m^3}{t}\log S)$, resulting in the running time

$$\begin{aligned}
&O(\frac{m^3}{t}\log S) \\
&\qquad = O(\frac{m^3 \cdot S}{dM}\log S) && \text{since } t = dM/S \\
&\qquad = O(\frac{m^3 \cdot S}{M}\log S) && \text{since } d \geq n-2 \\
&\qquad = O(\frac{m^3 \cdot mM}{M}\log S) && \text{since } mM \geq S \\
&\qquad = O(m^4 \log S).
\end{aligned}$$

- for $2-t-$Way-MinMax$-$Eq$-$CC: $L \leq \lceil(1 + \frac{2m}{t/4m^2})\log_2(2S)\rceil = \lceil(1 + \frac{8m^3}{t})\log_2(2S)\rceil$, resulting in the running time $O(m^4 \log S)$.

Therefore, Algorithm 2 running time is $O(m^4 \log S)$.

F Experiments

What is the effect of the number of split items s on the quality of the optimal partition in $n{-}s{-}\mathsf{Split}{-}\mathsf{MinMax}(\mathcal{X})$? We explored this question using experiments on random instances. Since the problem is NP-hard for $s < n-2$, we used a heuristic algorithm that we describe next.

F.1 An algorithm for every n and s

Theorem 4.2 implies that we can solve any instance of $n{-}s{-}\mathsf{Split}{-}\mathsf{MinMax}(\mathcal{X})$ in the following way (see Algorithm 3 in Appendix G for further details):

1. Let $\mathcal{X}_1 :=$ the s largest items in $\mathcal{X}$. Let $\mathcal{X}_2 :=$ the remaining items.
2. Solve $n{-}\mathsf{Way}{-}\mathsf{MinMax}(\mathcal{X}_2)$.
3. Split the items in $\mathcal{X}_1$ among the bins such that the resulting partition is as equal as possible.

While $n{-}\mathsf{Way}{-}\mathsf{MinMax}$ is NP-hard, there are several heuristic algorithms that can solve medium-sized instances in reasonable time [19]. We are interested in algorithms based on state-space search, such as the Complete Greedy Algorithm (CGA) [11]. CGA searches through all partitions by adding to each bin the largest number not yet added to any bin, so that bins with smaller sums are prioritized. This is done depth-first, meaning that the smallest of the input numbers are shuffled between different parts before larger input numbers are. In addition to the heuristics that make CGA run fast in practice, we add a new heuristic, specific to our setting: we stop the search whenever it finds a solution in which the maximum sum is at most $S := (\sum_{x\in X} x)/n$. This is because, by Lemma 4.1, we can then divide the s split items among the bins and get a perfect partition, in which every bin sum equals S.

F.2 Experiments

In Figure 1 we show that the possibility of splitting brings the optimal partition closer to the perfect partition, even when s is much smaller than $n-1$. We ran several instances of the $n{-}s{-}\mathsf{Split}{-}\mathsf{MinMax}(\mathcal{X})$ optimization problem on random inputs drawn from an exponential distribution, and from a uniform distribution with values between rM and M, for $r \in \{0, 0.5, 0.9, 0.99\}$. The number of items was $m \in \{10, 13, 15\}$, and number of bits in the item sizes was 16 or 32. Finally, the number of bins was between 2 and 10, each one represented by one curve in Figure 1. We ran seven different instances of each combination and plotted the mean of all instances. The x-axis of the plot represents the number of split items, the y-axis denotes the difference between the optimal and the perfect partition in percent: $100 \cdot (OPT - S)/S$.

G A fast algorithm for $n{-}s{-}\mathsf{Split}{-}\mathsf{MinMax}(\mathcal{X})$

In this section, we present a fast practical algorithm —Algorithm 3 — which we used to solve $n{-}s{-}\mathsf{Split}{-}\mathsf{MinMax}$ in appendix F for our experiments. It uses the following variant of Complete Greedy Algorithm (CGA): for any input list Y and upper bound S, it finds an n-way partition of Y where the maximum bin sum is at most S; if such a partition does not exist, it finds a partition that minimizes the maximum bin-sum. While its worst-case run-time is $O(2^m)$, it runs fast on practical instances [11]. Note that other practical algorithms

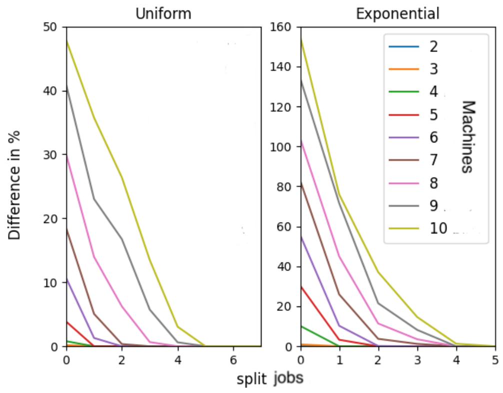

Figure 1: Difference between the optimal and the perfect partition, in percent. The plots shows results for uniform distribution with $r = 0$ (left plot) and exponential distribution (right plot). In both plots $m = 13$ and the item sizes have 16 bits. The results for other values of the parameters are qualitatively similar; we present them in Appendix L.

for optimal number partitioning could also be used. Denote the n-way partition obtained by the CGA for a given Y and a given upper bound S by CGA(Y, S).

Algorithm 3 n–s–Split–MinMax

```
 1: Order the items such that $x_1 \geq x_2 \geq \cdots \geq x_m$
 2: $\mathcal{X}_1 \longleftarrow \{x_1, \ldots, x_s\}$        ▷ $s$ largest items in $\mathcal{X}$
 3: $A \longleftarrow \sum_{x \in \mathcal{X}_1} x$
 4: $\mathcal{X}_2 \longleftarrow X \setminus \mathcal{X}_1$.
 5: $P_2 \longleftarrow CGA(\mathcal{X}_2, S)$        ▷ optimal (min-max) $n$-way partition of $\mathcal{X}_2$
 6: The bin sums in $P_2$ are $b_1, \ldots, b_n$, and we order the bins such that $b_1 \leq \cdots \leq b_n$
 7: for $i \longleftarrow 1$ to $n - 1$ do
 8:     $B \longleftarrow i \cdot (b_{i+1} - b_i)$        ▷ $B \geq 0$ since $b_{i+1} \geq b_i$
 9:     if $A < B$ then        ▷ Not enough split items to attain $b_{i+1}$
10:         Break
11:     else        ▷ Use split items to increase $i$ lowest bins to $b_{i+1}$
12:         $A \longleftarrow A - B$
13:         $b_1, \cdots, b_i \longleftarrow b_{i+1}$
14:     end if
15: end for
16: $b_1, \cdots, b_i \longleftarrow b_i + A/i$
```

Based on Lemma 4.1, we may assume that only the s largest items are split. We remove these items from the instance, and keep their sum in the variable A. Then we compute a min-max partition of the remaining items. Then, we try to split the amount A among the bins, such that the maximum bin sum remains as small as possible. In each iteration i, we use some of the split items for increasing the sums in the i lowest-sum such that their sum becomes equal to the $(i + 1)$-th bin. Finally, we divide the remaining amount of split items equally among the lowest-sum bins.

H A greedy algorithm for n–t–$\mathsf{Interval}(\mathcal{X})$ when $t \geq 1$

Given an instance of n–t–$\mathsf{Interval}(\mathcal{X})$, we first check if there is an item of size larger than $(1+t)S$. If there is such an item, then obviously no partition is t-feasible, so we answer "no".

Suppose now that all item sizes are at most $(1+t)S$. We show a greedy algorithm that always finds a t-feasible partition (so the answer is always "yes").

Because the total size of all items is nS, there are at most n items with size S or greater. We put each of them into its own bin. Now, only the items with size less than S remain.

The total extra space in the bins (when compared to the total size of all items) is $tS \cdot n \geq nS$, since $t \geq 1$. Therefore, at each moment, at least one bin always has S or more free space available. We take one of the remaining items, and put it into one such bin arbitrarily. We repeat this argument until all remaining items are allocated.

To summarize, when $t \geq 1$, the answer to n–t–$\mathsf{Interval}(\mathcal{X})$ is "yes" if all item sizes are at most $(1+t)S$, and "no" otherwise. This can be decided in time $O(m)$.

I Simple example for Lemma 3.3

In this section we show an instance with a solution with two almost-full bins, and no solution with at most one almost-full bin. Set $n = 5,\ d = n-2 = 3,\ \mathcal{X} = (n+2)\cdot[1] = (1,1,1,1,1,1,1)$. Then, with this setting, $m = 7,\ 5M = 1,\ 5S = 7,\ S + dM = 7/5 + 3/5 = 2$. Now, we can compute t and ϵ: $t = dM/S = (3/5)/(7/5) = 3/7$ and $\epsilon = t/(4 \cdot m^2) = 3/1372$. That is, an almost-full bin is a bin with sum larger that $(1+t)\cdot S/(1+\epsilon) = ((10/7)\cdot(7/5))/(1375/1372) = 2/(1375/1372) = 2744/1375 < 2$. Also, a d-feasible solution of the instance is a solution where the largest bin sum is at most $(1+t)S = 2$. Therefore, one possible partition is $P_1 = ((1,1),(1,1),(1),(1),(1))$. This partition is d-possible since the largest bin sum equals 2 as required. It has two almost-full bins: #1 and #2. Note that there is also a partition with three almost-full bins: $((1,1),(1,1),(1,1),(1),())$. However, there is no solution with at most one almost-full bin. Let us see that P_1 satisfies all the properties in the structure Lemma.

(a) Exactly two bins are almost-full: bin #1 and bin #2.

(b) The sum of bins 3,4,5 (not-almost-full) satisfies

$$\left(1 - \frac{2}{n-2}t - 2\epsilon\right)\cdot S \leq b_3, b_4, b_5 \leq \left(1 - \frac{2}{n-2}t + (n-1)2\epsilon\right)\cdot S$$
$$(1 - 2/7 - 3/686)\cdot 7/5 \leq b_3, b_4, b_5 \leq (1 - 2/7 + 6/342)\cdot 7/5$$
$$0.99387755102 \leq b_3, b_4, b_5 \leq 1.02456140351,$$

which is true since $b_3 = b_4 = b_5 = 1$

(c) Every almost-full bin contains only *big-items*, items largest than $nS(\frac{t}{n-2} - 2\epsilon) = 7(1/7 - 3/686) = 0.9693877551$. Since every item equals 1, the claim holds.

(d) Every not-almost-full bin contains big-items that are larger or equal than every item in bin 1,2, or *small-items*. Since all items are big and have the same size, the claim holds trivially.

(e) There are no *medium-items* at all since every item is a big-item.

(f) Every not-almost-full bin contains the same number of big-items, $\ell = 1$.

(g) Every almost-full bin contains $\ell + 1 = 2$ big-items.

Keeping this settings, we can increase or decrease the number of bins, recompute every parameter according to the new n, and reach the same conclusion. We built a generic example, such that if we modify n, all the other parameters can be recomputed because they are only dependent on n. For example, for $n = 4$, we will have $d = n - 2 = 4, \mathcal{X} = (n + 2) \cdot [1] = [1, 1, 1, 1, 1, 1]$. Therefore, we reach the same conclusions with the solution $[1, 1], [1, 1], [1], [1]$.

J Properties of instances with at least two almost-full bins

In this section we provide a detailed proof of Lemma 3.3.

Recall that, by the lemma assumption, there exists a t-feasible partition with some $r \geq 2$ almost-full bins $1, \ldots, r$. We take an arbitrary such partition and convert it using a sequence of transformations to another t-feasible partition satisfying the properties stated in the lemma, as explained below.

Proof of Lemma 3.3, property (a). Pick a t-feasible partition with some $r \geq 3$ almost-full bins. Necessarily $r \leq n - 1$, otherwise the sum of n almost-full bins would be larger than nS. It is sufficient to show that there exists a t-feasible partition with $r - 1$ almost-full bins; then we can proceed by induction down to 2. Assume w.l.o.g. that the almost-full bins are $1, \ldots, r$ and the not-almost-full bins are $r + 1, \ldots, n$. By definition,

$$b_1, \ldots, b_r \geq (1 + t) \cdot S/(1 + \epsilon) > ((1 + t) \cdot S)(1 - \epsilon) > (1 + t - 2\epsilon) \cdot S,$$

so,

$$\sum_{i=r+1}^{n} b_i = nS - \sum_{i=1}^{r} b_i < nS - rS \cdot (1 + t - 2\epsilon) = (n - r - rt + 2r\epsilon) \cdot S.$$

Denote by $b_{\min}$ the bin with the smallest bin sum (breaking ties arbitrarily). By the pigeonhole principle,

$$b_{\min} < \frac{(n - r - rt + 2r\epsilon) \cdot S}{n - r} = \left(1 - \frac{r}{n - r}t + \frac{r}{n - r}2\epsilon\right) \cdot S.$$

By assumption, all item sizes are at most nM. Suppose we take one item from bin r, and move it to $b_{\min}$. Then after the transformation,

$$\begin{aligned}
b_{\min} &< \left(1 - \frac{r}{n - r}t + \frac{r}{n - r}2\epsilon\right) \cdot S + nM \\
&\leq \left(1 - \frac{3}{n - 3}t + \frac{n - 1}{n - (n - 1)}2\epsilon\right) \cdot S + ntS/d && (3 \leq r \leq n - 1 \\
& && \text{and } dM = tS) \\
&\leq \left(1 - \frac{3}{n - 3}t + 2n\epsilon - 2\epsilon\right) \cdot S + \frac{n}{n - 2}tS && (d \geq n - 2) \\
&< \left(1 - \frac{2}{n - 2}t - 2\epsilon\right) \cdot S + \frac{n}{n - 2}tS && (\epsilon = t/4m^2) \\
&= (1 + t - 2\epsilon) \cdot S < (1 + t) \cdot S/(1 + \epsilon),
\end{aligned}$$

so $b_{\min}$ remains not-almost-full.

If bin r is still almost-full, then we are in the same situation: we have exactly r almost-full bins. So we can repeat the argument, and move another item from bin r to the (possibly different) minimum-sum bin. Since the number of items in bin r decreases with each step, eventually, it becomes not-almost-full. $\square$

This process continues until we have a partition with exactly two almost-full bins, which we assume to be bins 1 and 2. The not-almost-full bins are bins $3, \dots, n$. We now transform the partition so that the sum in each bin $3, \dots, n$ is bounded within a small interval.

Proof of Lemma 3.3, property (b). For proving the lower bound, suppose there is some $i \in \{3, \dots, n\}$ with $b_i < \left(1 - \frac{2}{n-2}t - 2\epsilon\right) \cdot S$. Move an item from bin 1 to bin i. Its new sum satisfies

$$\begin{aligned} b_i < \left(1 - \frac{2}{n-2}t - 2\epsilon\right) \cdot S + nM &= \left(1 - \frac{2}{n-2}t - 2\epsilon\right) \cdot S + ntS/d \\ &= \left(1 + \left(-\frac{2}{n-2} + \frac{n}{d}\right) t - 2\epsilon\right) \cdot S \\ &\leq \left(1 + \left(\frac{-2}{n-2} + \frac{n}{n-2}\right) t - 2\epsilon\right) \cdot S \\ &= (1 + t - 2\epsilon) \cdot S, \end{aligned} \tag{3}$$

so bin i is still not almost-full. Assumption (1) implies that b_1, b_2 must still be almost-full. So we are in the same situation and can repeat the argument until the lower bound is satisfied.

The upper bound on b_i is proved by simply subtracting the lower bounds of the other $n-1$ bins from the sum of all items, nS:

$$\begin{aligned} b_i &\leq nS - (b_1 + b_2) - (n-3)\left(1 - \frac{2}{n-2}t - 2\epsilon\right) \cdot S \\ &< nS - 2((1 + t - 2\epsilon) \cdot S) - (n-3)\left(1 - \frac{2}{n-2}t - 2\epsilon\right) \cdot S \\ &= \left(n - 2(1 + t - 2\epsilon) - (n-3)\left(1 - \frac{2}{n-2}t - 2\epsilon\right)\right) \cdot S \\ &= \left(1 - \frac{2}{n-2}t + (n-1)2\epsilon\right) \cdot S, \end{aligned} \tag{4}$$

so property (b) is satisfied. $\square$

A corollary of property (b) is that the difference between the sums of two not-almost-full bins is at most $2n\epsilon S = nSt/2m^2$.

Next, we transform the partition such that the almost-full bins 1 and 2 contain only "big" items, which we defined as items larger than $nS\left(\frac{t}{n-2} - 2\epsilon\right)$.

Proof of Lemma 3.3, property (c). Suppose bin 1 or 2 contains an item of size at most $nS\left(\frac{t}{n-2} - 2\epsilon\right)$. We move it to some not-almost-full bin b_i, $i \geq 3$. By property (b), the new sum of bin i satisfies

$$\begin{aligned} b_i &\leq \left(1 - \frac{2}{n-2}t + (n-1)2\epsilon\right) \cdot S + nS\left(\frac{t}{n-2} - 2\epsilon\right) \\ &= \left(1 + \left(-\frac{2}{n-2} + \frac{n}{n-2}\right) t + (n-1-n)2\epsilon\right) \cdot S \\ &= (1 + t - 2\epsilon)S, \end{aligned}$$

so bin i remains not-almost-full. Assumption (1) implies that b_1, b_2 remain almost-full. So properties (a), (b) still hold, and we can repeat the argument. Each move increases the sum in the not-almost-full bins, so eventually all items of size at most $nS(\frac{t}{n-2} - 2\epsilon)$ are in bins $3, \dots, n$. □

Next, we transform the partition so that the not-almost-full bins contain only the largest big-items and all the small-items.

Proof of Lemma 3.3, property (d). Suppose there exists some $i \in \{3, \dots, n\}$ for which b_i contains an item x such that $x \geq 2nS\epsilon$, but $x < y$ for some item y in bin 1 or 2. We exchange x and y. The sum in bin i increases, but not too much:

$$\begin{aligned} b_i &\leq \left(1 - \frac{2}{n-2}t + (n-1)2\epsilon\right) \cdot S + nM - 2nS\epsilon \\ &\leq \left(1 - \frac{2}{n-2}t + (n-1)2\epsilon\right) \cdot S + \frac{ntS}{d} - 2nS\epsilon && \text{(since } dM = tS\text{)} \\ &\leq \left(1 - \frac{2}{n-2}t + (n-1)2\epsilon\right) \cdot S + \frac{ntS}{n-2} - 2nS\epsilon && \text{(since } d \geq n-2\text{)} \\ &\leq (1 + t - 2\epsilon) \cdot S, \end{aligned}$$

so b_i is sill not almost-full. So by (1), bins 1 and 2 must remain almost-full, and we can repeat the argument. Each move increases the sum in bins $3, \dots, n$, so the process must end. □

Properties (c) and (d) imply that the input list $\mathcal{X}$ contains no medium items; for completeness and verification, we provide a stand-alone proof below.

Proof of Lemma 3.3, property (e). Assume that there is an allocation in which bins 1 and 2 are almost-full, and bins $3, \dots, n$ are not-almost-full. Suppose for contradiction that $\mathcal{X}$ contains a medium item x. Consider two cases.

Case 1: x is in an almost-full bin, say bin 1. Then, we can move x to some not-almost-full bin $i \geq 3$. By the inequalities in the proof of (c), after the move we have $b_i < (1 + t - 2\epsilon)S$, so bin i remains not-almost-full. Moreover, since the previous sum of bin 1 was at most $(1 + t)S$, its new sum is: $b_1 < (1 + t)S - (2nS\epsilon) = (1 + t - 2n\epsilon)S < (1 + t - 2\epsilon)S$, since $n > 1$. So bin 1 is not almost-full anymore. We now have a new allocation with only one almost-full bin (bin 2). Similarly, if x is in bin 2, we can move it to another bin, and get an allocation with only bin 1 almost-full. This contradicts the lemma assumption (1).

Case 2: x is in some not-almost-full bin $i \geq 3$. We exchange x with some big item y in bin 1. By the inequalities in the proof of (d),

$$b_i < (1 + t - 2\epsilon) \cdot S,$$

so bin i remains not-almost-full. The sum in bin 1 decreases due to the exchange. If bin 1 becomes not almost-full, then we have a new allocation with only bin 2 almost-full. If bin 1 remains almost-full, then we have a situation as in Case 1; by moving x to some bin $i \geq 3$, we get a new allocation with only bin 2 almost-full. In both cases, this contradicts (1). □

Proof of Lemma 3.3, property (f). We first claim that, for every two big-items, x, y, the difference $|y - x| < 2nS\epsilon$. Assume w.l.o.g that $y > x$. By definition, nM is the largest item size, so $y \leq nM$. Since x is a big-item, $x > nS\left(\frac{t}{n-2} - 2\epsilon\right)$. So

$$\begin{aligned}
y - x &< nM - nS\left(\frac{t}{n-2} - 2\epsilon\right) \\
&= \frac{ntS}{d} - \frac{ntS}{n-2} + 2nS\epsilon && \text{(since } dM = tS\text{)} \\
&\leq 2S\epsilon n. && \text{(since } d \geq n-2\text{)}
\end{aligned}$$

Now, assume for contradiction that two not-almost-full bins contain a different number of big-items; say bin j has at least one big-item more than bin i. We show that even if all the big-items in bin j are smaller than all big-items in bin i, and all the (at most m) small-items are in bin i, the difference between them satisfies

$$\begin{aligned}
b_j - b_i &> nS\left(\frac{t}{n-2} - 2\epsilon\right) - m \cdot 2n\epsilon S - m \cdot 2n\epsilon S \\
&= nS\left(\frac{t}{n-2} - 2\epsilon\right) - 2m \cdot 2n\epsilon S \\
&= nS\left(\frac{t}{n-2} - 2\epsilon(1 + 2m)\right) \\
&\geq nS\left(\frac{t}{m-2} - \frac{t}{m}\right) && \text{(since } \epsilon = t/4m^2\text{)} \\
&= nS\left(\frac{2t}{(m-2)m}\right) \\
&> 2nSt/m^2.
\end{aligned}$$

But by property (b), the difference is at most $2nS\epsilon = nSt/2m^2$ — a contradiction. $\square$

Proof of Lemma 3.3, property (g). The size of every item is at most $nM \leq ntS/(n-2)$. By (f), each not-almost-full bin i contains ℓ big-items. So by the previous conditions, the size of not-almost-full bin is

$$b_i \leq \ell \cdot ntS/(n-2) + m \cdot 2n\epsilon S \tag{5}$$

We now focus on almost-full bin 1; the proof for bin 2 is the same. If bin 1 contains fewer than $\ell + 1$ items, then $b_1 \leq b_i$, which contradicts (1).

Assume for contradiction that bin 1 contains more than $\ell + 1$ items. Then:

$$\begin{aligned}
b_1 &\geq (\ell + 2) \cdot \left(nS\left(\frac{t}{n-2} - 2\epsilon\right)\right) && \text{(by big-item size)} \\
&= \ell nSt/(n-2) + m \cdot 2n\epsilon S - m \cdot 2n\epsilon S \\
&\qquad - \ell nS2\epsilon + 2nSt/(n-2) - 4nS\epsilon \\
&\geq b_i - m \cdot 2n\epsilon S - \ell nS2\epsilon + 2nSt/(n-2) - 4nS\epsilon && \text{(by 5)} \\
&\geq \left(1 - \frac{2}{n-2}t - 2\epsilon\right) \cdot S - m \cdot 2n\epsilon S - \ell nS2\epsilon \\
&\qquad + 2nSt/(n-2) - 4nS\epsilon && \text{(by 3)} \\
&= S + \left(\frac{2n-2}{n-2}\right) tS - m \cdot 2n\epsilon S - \ell nS2\epsilon - 4nS\epsilon - 2\epsilon S \\
&= S + tS + tS((n/n-2) - m \cdot 2n\epsilon/t - \ell n2\epsilon/t - 4n\epsilon/t - 2\epsilon/t) \\
&> S + tS + tS(1 - 1/2 - \ell/2m - 1/m - 1/2nm) && (n < m \text{ so } \epsilon < t/4nm) \\
&> S + tS, && \text{(since } 0 < \ell < m/3\text{)}
\end{aligned}$$

which contradicts the assumption that $(b_1, b_2, \ldots, b_n)$ represent a t-feasible partition. □

K n−s−Split−MinMax

In the section, we explain the binary search procedure used to solve the n−s−Split−MinMax ($\mathcal{X}$) optimization problem. We first prove a property of the optimal bin sum of the n−s−Split −MinMax($\mathcal{X}$) problem.

Lemma K.1. *If the optimal value of an n−s−Split−MinMax($\mathcal{X}$) instance is not an integer, then the optimal partition of $\mathcal{X}$ is a perfect partition (that is, all bin sums are equal to the average bin size S).*

Proof. Let OPT be the maximum bin sum in an optimal partition of $\mathcal{X}$ with s split items. If OPT is not an integer, then the fractional part of the bin sum must come from a split item (since the non-split items have integer sizes). Since the partition is not perfect, there is a bin with sum smaller than OPT. Move a small fraction of the split item from all bins with sum OPT to a bin with sum smaller than OPT. This yields a partition with a smaller maximum bin sum — a contradiction to optimality. □

Now, we show how to use binary search in order to solve the n−s−Split−MinMax($\mathcal{X}$) problem. The idea is to execute the n−s−t−Split−Bound($\mathcal{X}$) algorithm several times, trimming the parameter t at each execution in a binary search style. When $t = n - 1$, the answer to n−s−t−Split−Bound($\mathcal{X}$) is always "yes", since even if we put all items in a single bin, its sum is $nS = (1 + t)S$. Moreover, by Lemma K.1, the optimal bin sum value of n−s−Split −MinMax($\mathcal{X}$) is either an integer or equal to S. Therefore, in addition to considering the case where $t = 0$, we only need to consider values of t that differ by at least $1/S$, since if the bin sums are not equal to S they are always integers, so they differ by at least 1. Therefore, at most nS different values of t have to be checked, so the binary search requires at most $\log_2(nS)$ executions of n−s−t−Split−Bound($\mathcal{X}$). This is polynomial in the size of the binary representation of the input.

L Additional Plots

We present the plots obtained for different values of the parameters presented in the Appendix F. All the plots behave similarly, showing that the difference between the optimal and the perfect partition decreases when the number of split items increases.

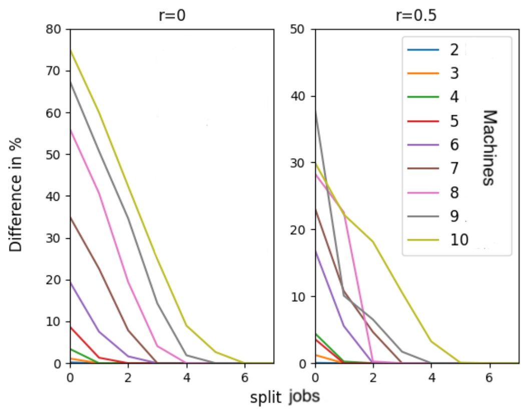

Figure 2: Uniform distribution with $m = 10$ and 16 bits.

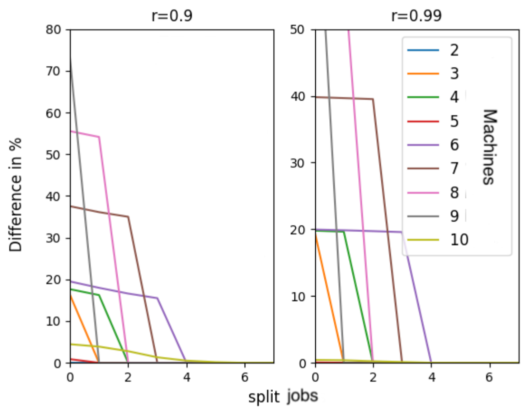

Figure 3: Uniform distribution with $m = 10$ and 16 bits.

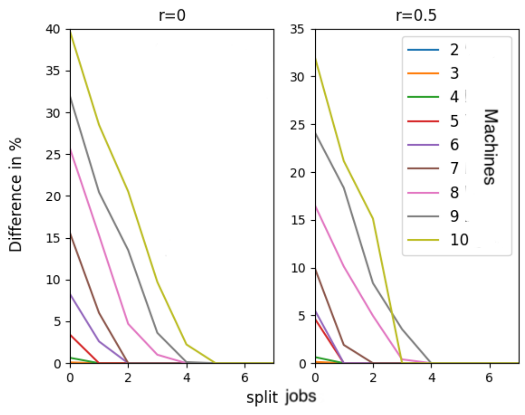

Figure 4: Uniform distribution with $m = 13$ and 16 bits.

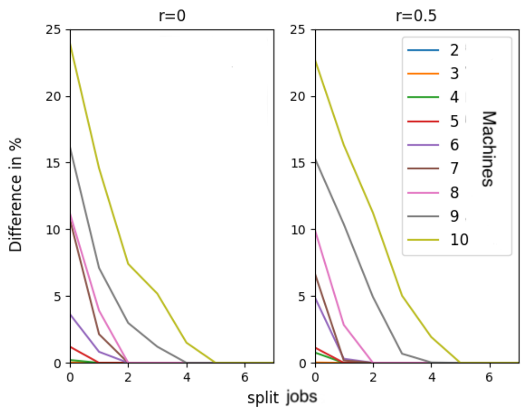

Figure 5: Uniform distribution with $m = 15$ and 16 bits.

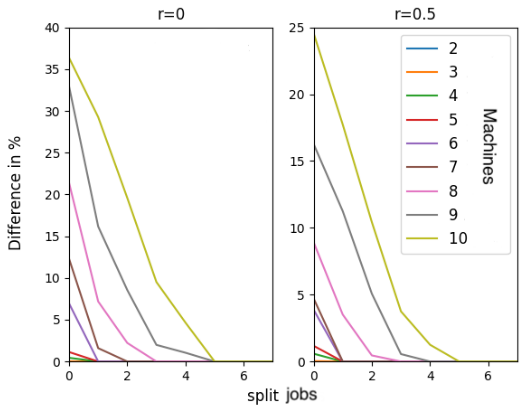

Figure 6: Uniform distribution with $m = 15$ and 32 bits.

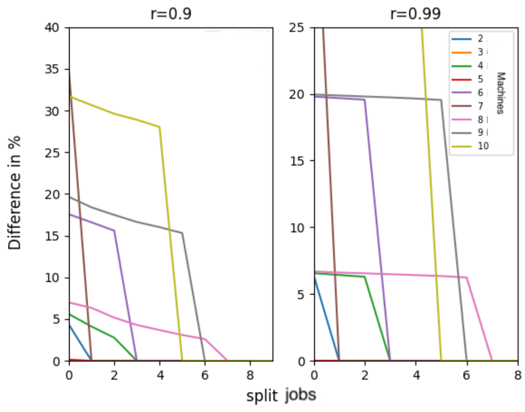

Figure 7: Uniform distribution with $m = 15$ and 32 bits.

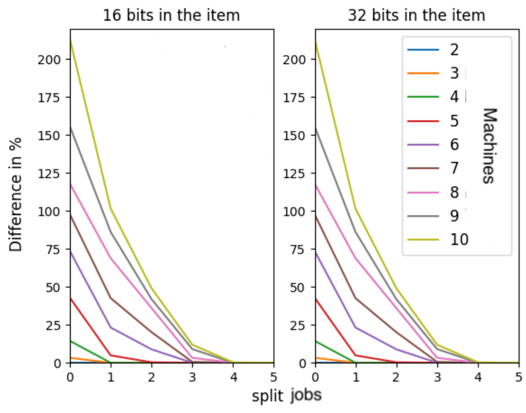

Figure 8: Exponential distribution with $m = 10$.

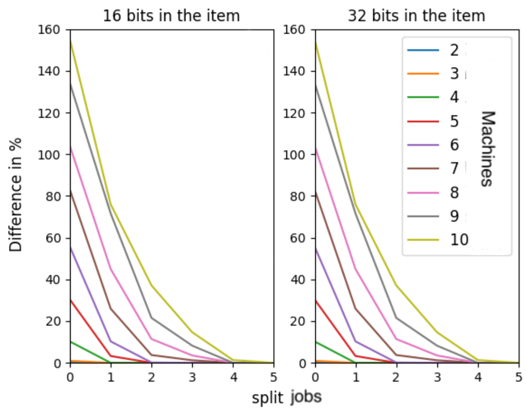

Figure 9: Exponential distribution with $m = 13$.

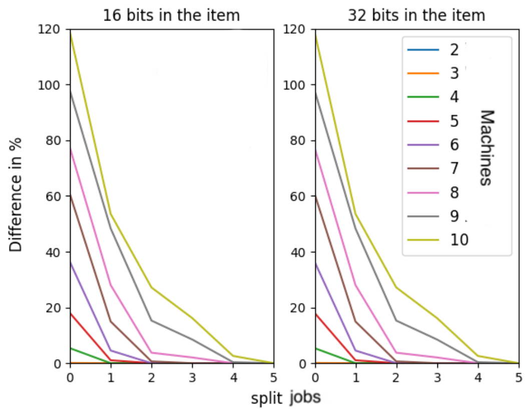

Figure 10: Exponential distribution with $m = 15$.