A Simple and Fast Linear-Time Algorithm for Divisor Methods of Apportionment*

Raphael Reitzig† Sebastian Wild*

January 18, 2023

Proportional apportionment is the problem of assigning seats to states (resp. parties) according to their relative share of the population (resp. votes), a field heavily influenced by the early work of Michel Balinski, not least his influential 1982 book with Peyton Young [BY01].

In this article, we consider the computational cost of *divisor methods* (also known as *highest averages* methods), the de-facto standard solution that is used in many countries. We show that a simple linear-time algorithm can exactly simulate all instances of the family of divisor methods of apportionment by reducing the problem to a single call to a selection algorithm. All previously published solutions were iterative methods that either offer no linear-time guarantee in the worst case or require a complex update step that suffers from numerical instability.

1. Introduction

The mathematical problem of *proportional apportionment* arises whenever we have a finite supply of k indivisible, identical resources which are to be distributed across n parties proportionally to their publicly known and agreed-upon values $v_1, \dots, v_n$. The indivisibility constraint makes a perfectly proportional assignment impossible unless the *quotas* $k \cdot v_i/V$ with $V = v_1 + \cdots + v_n$ happen to be all integral for $i = 1, \dots, n$; apportionment methods decide how to allocate resources in the general case.

Apportionment directly arises in politics in two forms:

- In a proportional-representation electoral system seats in parliament are assigned to political parties according to their share of all votes. (The resources are seats, and the values are vote counts.)

*Most of this work was done while both authors were at University of Kaiserslautern.
†Department of Computer Science, University of Kaiserslautern; {reitzig, wild}@cs.uni-kl.de

- In federal states the number of representatives from each component state often reflects the population of that state. (Resources are again seats, values are the numbers of residents.)

While not identical in their requirements – for example, any state will typically have at least one representative no matter how small it is – the same mathematical framework applies to both instances. Further applications are tables wherein rounded percentages should add up to 100%, the assignment of workers to jobs, or the allocation of service facilities to areas proportional to demand.

In order to use consistent language throughout this article, we will stick to the first metaphor. That is, we assign k *seats* to n *parties* proportionally to their respective *votes* v_i; we call k the *house size*. In the case of electoral systems which exclude parties below a certain threshold of overall votes from seat allocation altogether, we assume they have already been removed from our list of n parties. An *apportionment method* maps vote counts $\mathbf{v} = (v_1, \ldots, v_n)$ and house size k to a seat allocation $\mathbf{s} = (s_1, \ldots, s_n)$ so that $s_1 + \cdots + s_n = k$. We interpret $\mathbf{s}$ as party i getting s_i seats.

There are many conceivable such methods, but Balinski and Young [BY01] show that the *divisor methods* (introduced below) are the only methods that guarantee *pairwise vote monotonicity* (*population monotonicity* in [BY01]), which requires that a party i cannot *lose* seats to a party j when i *gains* votes while j loses votes (and all other parties remain unchanged). For a comprehensive introduction into the topic with its historical, political, and mathematical dimensions, including desirable and undesirable properties of various apportionment methods and corresponding impossibility results, we refer the reader to the books of Balinski and Young [BY01] and Pukelsheim [Puk14].

1.1. Problem Definition

Divisor methods (also known as *highest-averages methods*) are characterized by the used *rounding rule* $[\![\cdot]\!]$; examples include rounding down, rounding up, or rounding to the nearest even integer (see also Table 1 and [Puk14, p. 70]). Party i is then assigned $[\![v_i/D]\!]$ seats, where D is a *divisor* chosen so that $s_1 + \cdots + s_n = k$; such a D is guaranteed to exist for any sensible rounding rule and obtained by solving the following optimization problem: $\max D$ s.t. $\sum_{i=1}^{n} [\![v_i/D]\!] \geq k$. We point out that without an algorithm to solve this problem, divisor methods of apportionment cannot feasibly be applied in practice.

While the concept of divisor methods can be used more generally, a typical assumption is that $\lfloor x \rfloor \leq [\![x]\!] \leq \lceil x \rceil$, which implies that s_i is roughly proportional to v_i/V. For this section, we also make this assumption; later, we slightly weaken it to nearly-arithmetic divisor sequences (Definition 1).

Different rounding rules yield in general different apportionment methods, and there is no *per se* best choice. For example, there are competing notions of fairness, each favoring a different divisor method [BY01, Section A.3]. A reasonable approach is therefore to run computer simulations of different methods and compare their outcomes empirically, for example w.r.t. the distribution of final average votes per seat v_i/s_i. For this purpose, many

apportionments may have to be computed, making efficient algorithms desirable. Apart from that, settling the computational complexity of this fundamental optimization problem is interesting in its own right.

1.2. Previous Work

While methods for proportional apportionment have been studied for a long time, the question of algorithmic complexity has only more recently been considered. A direct iterative method (see Section 2) has complexity $\Theta(nk)$ when implemented naively, and $\Theta(k \log n)$ when using a priority queue. Note that typically $k \gg n$, and indeed, the input consists of $n+1$ numbers, so this running time can be exponential in the size of a binary encoding of the input.

A simple refinement, the jump-and-step algorithm described by Pukelsheim [Puk14] (see also Section 4.1), avoids any dependency on k. It is based on the iterative method, but jumps to within $O(n)$ of the target value, so worst-case running times are $O(n^2)$ with naive iteration and $O(n \log n)$ using a priority queue. These bounds seem to be folklore; they are mentioned explicitly for example by [DK99; Zac06]. This running time is not optimal, but the algorithm is simple and performs provably well in certain average-case scenarios [Puk14, §6.7].

Finally, Cheng and Eppstein [CE14] obtained an algorithm with the optimal $O(n)$ complexity in the worst case. They reduce the problem of finding a divisor D to selecting the kth smallest element from a multiset formed by n arithmetic progressions, and design a somewhat involved algorithm to solve this special rank-selection problem in $O(n)$ time. This settles the theoretical complexity of the problem since clearly $\Omega(n)$ time is necessary to read the input. However, apart from conceptual complexity, Cheng and Eppstein's algorithm suffers from a numerical-instability issue that we uncovered when implementing their algorithm.

1.3. Contribution

Our main contribution is a much simpler algorithm than Cheng and Eppstein's algorithm for divisor methods of apportionment. It directly constructs a multiset $\hat{\mathcal{A}}$ of size $O(n)$ and a rank $\hat{k}$ so that D is obtained as the $\hat{k}$th smallest element in $\hat{\mathcal{A}}$. An example execution of our algorithm is shown in Table 2 (page 8). Apart from its improved conceptual simplicity and practical efficiency (see Section 4), this also circumvents any issues from imprecise arithmetic. Formally, our result is as follows.

Theorem 1 (Main result):
Given any rounding rule $[\![\cdot]\!]$ with $\lfloor x \rfloor \leq [\![x]\!] \leq \lceil x \rceil$ for all $x \in \mathbb{R}_{\geq 0}$, any vector of votes $\mathbf{v} \in \mathbb{N}^n$, and house size $k \in \mathbb{N}$, our algorithm SandwichSelect *computes a divisor D that yields seat allocations $\mathbf{s} = (s_1, \ldots, s_n)$ respecting $s_i \in [\![\frac{v_i}{D}]\!]$ and $s_1 + \cdots + s_n = k$ using running time in $O(n)$. It can do so without explicitly computing $[\![\cdot]\!]$.*

Method	also known as	Divisor Sequence	$\delta(x)$	Sandwich
Smallest divisors	Adams	0, 1, 2, 3, ...	x	—
Greatest divisors	d'Hondt, Jefferson	1, 2, 3, 4, ...	$x+1$	—
Sainte-Laguë	Webster, Major Fractions	1, 3, 5, 7, ...	$2x+1$	—
Modified Sainte-Laguë		1.4, 3, 5, 7, ...	$\begin{cases} 2x+1 & x\geq 1 \\ 1.6x+1.4 & x<1 \end{cases}$	$2x+\frac{6}{5}\pm\frac{1}{5}$
Equal Proportions	Huntington-Hill	0, $\sqrt{2}$, $\sqrt{6}$, $\sqrt{12}$, ...	$\sqrt{x(x+1)}$	$x+\frac{1}{4}\pm\frac{1}{4}$
Harmonic Mean	Dean	0, $\frac{4}{3}$, $\frac{12}{5}$, $\frac{24}{7}$, ...	$\frac{2x(x+1)}{2x+1}$	$x+\frac{1}{4}\pm\frac{1}{4}$
Imperiali		2, 3, 4, 5, ...	$x+2$	—
Danish		1, 4, 7, 10, ...	$3x+1$	—

Table 1: Commonly used divisor methods [CE14, Table 1]. For each of the methods, we give a possible continuation δ of the respective divisor sequence as well as linear sandwich bounds on δ (where nontrivial; cf. Lemma 4 on page 10).

Moreover, we report from an extensive running-time study of the above apportionment methods. We find that our new method is almost an order of magnitude (a factor 10) faster than Cheng and Eppstein's algorithm while at the same time avoiding the super-linear complexity of the jump-and-step algorithm for large inputs. Implementations of all algorithms and sources for the experiments are available online [RW15].

Outline. Section 2 defines divisor methods formally. In Section 3, we describe the selection-based algorithms, including our new method. Section 4 describes results of our running-time study; Section 5 concludes the paper. For the reader's convenience, we include an index of notation in Appendix A.

2. Preliminaries

Our exposition follows the notation of Cheng and Eppstein [CE14], but we also give the names as used by Pukelsheim [Puk14].

2.1. Divisor Sequences

A *divisor sequence* is a nonnegative, strictly increasing and unbounded sequence of real numbers. Throughout the paper, we consider a fixed divisor sequence $d = (d_j)_{j=0}^{\infty}$; for notational convenience, we set $d_{-1} := -\infty$. We require a monotonic continuation δ of d on the reals which is easy to invert; formally, we assume a function $\delta : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq d_0}$ with

(D1) δ is continuous and strictly increasing,

(D2) $\delta^{-1}(x)$ for $x \geq d_0$ can be computed with a constant number of arithmetic operations, and

(D3) $\delta(j) = d_j$ (and thus $\delta^{-1}(d_j) = j$) for all $j \in \mathbb{N}_0$.

All the divisor sequences used in practice fulfill these requirements; cf. Table 1. For convenience, we continue δ^{-1} on the complete real line requiring

(D4) $\delta^{-1}(x) \in [-1, 0)$ for $x < d_0$.

Lemma 1: *Assuming (D1) to (D4), $\delta^{-1}(x)$ is continuous and strictly increasing on $\mathbb{R}_{\geq d_0}$. Furthermore, it is the inverse of $j \mapsto d_j$ in the sense that*

$$\lfloor \delta^{-1}(x) \rfloor = \max\{j \in \mathbb{Z}_{\geq -1} \mid d_j \leq x\}$$

for all $x \in \mathbb{R}$. □

In particular, $\lfloor \delta^{-1}(x) \rfloor + 1 = \big|\{j \in \mathbb{N}_0 : d_j \leq x\}\big|$ is the *rank* function for the set of all d_j.

In the terminology of Pukelsheim [Puk14], d is a *jumppoint sequence* (of a rounding rule, see below), but with a shift of indices (we start with d_0 instead of $s(1)$). A divisor sequence with $j \leq d_j \leq j + 1$ for all $j \in \mathbb{N}_0$ is called a *signpost sequence*[1].

While divisor methods can be defined for any nonnegative, strictly increasing and unbounded sequence, we will focus our attention on those with the following property.

Definition 1 (nearly arithmetic): *A divisor sequence $(d_j)_{j\in\mathbb{N}_0}$ resp. its continuation δ is called nearly arithmetic if there are constants $\alpha > 0$, $\underline{\beta} \in [0, \alpha]$, and $\overline{\beta} \geq 0$ so that*

$$\forall x \in \mathbb{R}_{\geq 0} \qquad \alpha x + \underline{\beta} \;\leq\; \delta(x) \;\leq\; \alpha x + \overline{\beta}.$$

Table 1 lists divisor sequences for common apportionment methods; all are nearly arithmetic. Further, any signpost sequence is trivially nearly arithmetic, including power-mean signposts [Puk14, §3.11–12] and geometric-mean signposts [DK99]. Nearly arithmetic sequences are also exactly the class of divisor sequences addressed by Cheng and Eppstein [CE14].

2.2. Ties, Rounding Rules, and Seat Allocations

Since the actual seat allocation $\mathbf{s}$ is not uniquely determined in case of ties, it is convenient to have a set-valued *rounding rule* in addition to the rank function. The rounding rule $[\![\cdot]\!]$ induced by divisor sequence d is defined by $[\![x]\!] = \lfloor\!\lfloor \delta^{-1}(x) \rfloor\!\rfloor + 1$, were $\lfloor\!\lfloor x \rfloor\!\rfloor = \{\lfloor x \rfloor\}$ if $x \notin \mathbb{Z}$ and $\lfloor\!\lfloor n \rfloor\!\rfloor = \{n - 1, n\}$ for $n \in \mathbb{Z}$. (The $+1$ is due to the index shift in divisor sequences; $\lfloor\!\lfloor \cdot \rfloor\!\rfloor$ is the natural extension of $\lfloor \cdot \rfloor$ that returns *both* limits at jump discontinuities). Note that we have $\lfloor x \rfloor \leq [\![x]\!] \leq \lceil x \rceil$ (for one of the possible values of $[\![x]\!]$ in case of ties) if and only if the jumppoint sequence is a signpost sequence, making these particularly natural choices

[1] Note that Pukelsheim [Puk14] additionally requires that signpost sequences do not touch both endpoints of the interval $[j, j + 1]$ ("left-right disjunction"). Our conditions (D1) to (D4) already imply this property.

for rounding rules. The *set of valid seat assignments* for given votes and house size is then given by

$$\mathcal{S}(\mathbf{v}, k) \;=\; \left\{ \mathbf{s} \in \mathbb{N}_0^n \;\middle|\; \sum_{i=1}^{n} s_i = k \;\wedge\; \exists D > 0.\; \forall\, i \in [n].\; s_i \in \left[\!\left[\frac{v_i}{D} \right]\!\right] \right\}. \tag{1}$$

2.3. Highest Averages

Divisor methods can equivalently be defined by an iterative method [BY01, Prop. 3.3]: Starting with no allocated seats, $s_i = 0$ for $i \in [n]$, we iteratively assign the next seat to a party with a currently "highest average", i.e., maximal v_i/d_{s_i}: a party with the most votes per seat. For technical reasons, it turns out to be much more convenient to work with *reciprocal* averages, i.e., assign the next seat to a party with minimal d_{s_i}/v_i (fewest current seats per vote). In case of ties, any choice leads to a valid seat allocation $\mathbf{s} \in \mathcal{S}(\mathbf{v}, k)$.

This iterative method does not yield an efficient algorithm, but it gives rise to to a key structural observation: the minimal quotients d_j/v_i are weakly increasing over subsequent iterations (by monotonicity of d), and we obtain the final (largest) quotient directly as

$$a^* \;=\; \max \left\{ \frac{d_{s_i - 1}}{v_i} \;\middle|\; i \in [n] \right\} \tag{2}$$

using the final seat assignment $\mathbf{s}$. The iterative method yields the same seat assignments as Equation (1) using $D = 1/a^*$ (cf. [Puk14, 59f]); to get the full set of all feasible assignments $\mathcal{S}(\mathbf{v}, k)$, one has to simulate all possibilities of breaking ties when selecting the next party to be awarded a seat.

3. Fast Apportionment through Selection

Worst-case optimal algorithms for divisor methods of apportionment exploit that the quotients d_j/v_i in the iterative method change monotonically: The final multiplier a^* is the *kth smallest* of all possible quotients d_j/v_i, and can hence be found directly using a selection algorithm [CE14]. The challenge is to suitably restrict the candidate set from which to select.

We need some more notation. Given a (multi)set $\mathcal{M}$ of (not necessarily distinct) numbers, we write $\mathcal{M}_{(k)}$ for the kth order statistic of $\mathcal{M}$, i.e., the kth smallest element (counting duplicates) in $\mathcal{M}$. For example, if $M = \{5, 8, 8, 8, 10, 10\}$, we have $M_{(1)} = 5$, $M_{(2)} = M_{(3)} = M_{(4)} = 8$, and $M_{(5)} = M_{(6)} = 10$. For given votes $\mathbf{v} = (v_1, \ldots, v_n) \in \mathbb{Q}_{>0}^n$, we define the sets

$$A_i \;:=\; \{ a_{i,j} \mid j = 0, 1, 2, \ldots \} \quad \text{with} \quad a_{i,j} := \frac{d_j}{v_i}$$

and their multiset union $\mathcal{A} := \biguplus_{i=1}^{n} A_i$. With that notation, we obtain that $a^* = \mathcal{A}_{(k)}$. We further define the *rank function* $r(x, \mathcal{A})$ as the number of elements in multiset $\mathcal{A}$ that are

no larger than x, that is

$$r(x, \mathcal{A}) \;:=\; \big|\mathcal{A} \cap (-\infty, x]\,\big| \;=\; \sum_{i=1}^{n} \big|\{a_{i,j} \in \mathcal{A} \mid a_{i,j} \le x\}\big|. \tag{3}$$

We write $r(x)$ instead of $r(x, \mathcal{A})$ when $\mathcal{A}$ is clear from context. In light of the optimization formulation, $\min a$ s.t. $r(a, \mathcal{A}) \ge k$, we call a value a *feasible* if $r(a, \mathcal{A}) \ge k$, otherwise it is *infeasible*. Feasible $a > a^*$ are called *suboptimal*.

Note that $\mathcal{A}$ is infinite, but $\mathcal{A}_{(k)}$ always exists since the terms $a_{i,j} = {}^{d_j}/_{v_i}$ are strictly increasing in j for all $i \in \{1, \ldots, n\}$.

3.1. Cheng and Eppstein's Algorithm

Cheng and Eppstein [CE14] devise an iterative method that maintains an approximation ξ of a^*. In each step, the method either (at least) halves the difference of $r(\xi)$ to k or it (at least) halves the number of parties still under consideration. By ensuring that the initial distance of $r(\xi)$ from k is $O(n)$, their algorithm terminates after $O(n)$ iterations. Each iteration selects the median of the set of $a_{i,j}$ closest to ξ for all remaining parties i; using a linear-time selection algorithm, this yields overall $O(n)$ time. More concretely, their algorithm, ChengEppsteinSelect, uses the following three steps.

(a) Identify *contributing sequences* and compute an initial *coarse solution* ξ, i.e., a value with rank $r(\xi) = k \pm O(n)$.
(The initial coarse solution is essentially our $\overline{a}$ as defined below.)

(b) Compute a *lower-rank coarse solution* ξ' with rank $r(\xi') \in [k - n..k]$ starting with ξ.

(c) Compute a^* starting at ξ'.

Each of the steps involves a variant of the iterative median-based algorithm sketched above.

Remark 1 (Precision issues): While implementing it for our running time study, we discovered the following shortcoming of ChengEppsteinSelect. After the median selection, one has to determine for how many parties i the closest $a_{i,j}$ to ξ yields *exactly* the new upper bound u; all but one of these have to be excluded and their number must be known precisely (see the computation of m in Algorithm 2 of [CE14]). This comes down to testing whether $d_j/v_i = d_{j'}/v_{i'}$ for various values of i, j, i', j'; with a naive implementation based on floating-point arithmetic, this cannot be done reliably. The situation is aggravated by the fact that such an implementation can return incorrect results without any obvious signs of failure.

To circumvent this issue, one can either work with exact (rational) arithmetic, which slows down comparisons during median selection considerably, or keep a mapping from quotients $a_{i,j}$ back to party i to check $d_j/v_i = d_{j'}/v_{i'}$ by testing $d_j v_{i'} = d_{j'} v_i$. The latter requires additional space and slows down swaps during median selection. We are not aware of a fully satisfactory solution to this issue.

3.2. Our Algorithm

Our algorithm relies on explicitly constructing a small *"slice"* $\mathcal{A} \cap [\underline{a}, \overline{a}]$ that contains a^*; we can then directly apply a rank-selection algorithm on this slice. We delay any detailed

	Party	ÖVP	SPÖ	Martin	FPÖ	GRÜNE	BZÖ	Total
	Votes	858 921	680 041	506 092	364 207	284 505	131 261	2 825 027
Step 1	$\underline{a} = 17/2\,825\,027 = 6.01764 \cdot 10^{-6}$, $\overline{a} = 23/2\,825\,027 = 8.14151 \cdot 10^{-6}$							
Step 3.1	$v_i \cdot \underline{a}$	5.1687	4.0922	3.0455	2.1917	1.7121	0.7899	
		$\underline{j} = 5$	$\underline{j} = 4$	$\underline{j} = 3$	$\underline{j} = 2$	$\underline{j} = 1$	$\underline{j} = 0$	
	$v_i \cdot \overline{a}$	6.9929	5.5366	4.1204	2.9652	2.3163	1.0687	
		$\overline{j} = 5$	$\overline{j} = 4$	$\overline{j} = 3$	$\overline{j} = 1$	$\overline{j} = 1$	$\overline{j} = 0$	
Step 3.2	$\hat{\mathcal{A}}$	6.9855	7.3525	7.9037	—	7.0298	7.6184	$\cdot 10^{-6}$
	$\hat{k}$	$17 - 5 - 4 - 3 - 2 - 1 - 0 = 2$						
Step 4	$\hat{\mathcal{A}}_{(\hat{k})}$	$a^* = 7.0298$						
safe seats ($\underline{j}$)		5	4	3	2	1	0	15
+ number $a_{i,j} \le a^*$		1	0	0	0	1	0	2
Seats	s_i	6	4	3	2	2	0	17

Table 2: Execution of SandwichSelect on the example input from [Puk14, §4.9], the 2009 European Parliament election in Austria with $n = 6$ parties competing for $k = 17$ seats. The divisor sequence used is $d_j = j + 1$, so we have $\alpha = 1$ and $\underline{\beta} = \overline{\beta} = 1$ and we obtain $\delta^{-1}(x) = x - 1$.
The final seat distribution for party i is obtained by summing the $\underline{j}$ value for that party and the number of terms $a_{i,j}$ contributed to $\hat{\mathcal{A}}$ that are no larger than a^*. Note that to determine the seat distribution, we can simply recompute $a_{i,j}$ for $\underline{j} \le j \le \overline{j}$ using the exact same computation; any imprecision in these computations are without consequence as long as they are deterministic and errors are small enough to not affect the relative order of terms in $\hat{\mathcal{A}}$.

justifications to Section 3.3 and first state our algorithm. An application of the algorithm to an exemplary input is given in Table 2.

Recall that we assume a fixed apportionment scheme with a nearly-arithmetic divisor sequence, i.e., $\alpha j + \underline{\beta} \le d_j \le \alpha j + \overline{\beta}$ (Definition 1).

Algorithm 1: SandwichSelect$_d(\mathbf{v}, k)$:

Step 1 Compute $\underline{a} := \max\{0, \alpha\frac{k}{V} - (\alpha - \underline{\beta})\frac{n}{V}\}$ and $\overline{a} := \alpha\frac{k}{V} + \overline{\beta}\frac{n}{V}$.

Step 2 Initialize $\hat{\mathcal{A}} := \emptyset$ and $\hat{k} := k$.

Step 3 For $i = 1, \ldots, n$ do:

Step 3.1 Compute $\underline{j} := \max\{0, \lceil\delta^{-1}(v_i \cdot \underline{a})\rceil\}$ and $\overline{j} := \lfloor\delta^{-1}(v_i \cdot \overline{a})\rfloor$.

Step 3.2 For all $j = \underline{j}, \ldots, \overline{j}$, add d_j/v_i to $\hat{\mathcal{A}}$.

Step 3.3 Update $\hat{k} := \hat{k} - \underline{j}$.

Step 4 Select and return the $\hat{k}$th smallest element of $\hat{\mathcal{A}}$.

The intuition behind bounds $[\underline{a}, \overline{a}]$ on a^* is to investigate the rank of a^* in the multiset $\mathcal{A}$ of

all candidates. Since any number between one and n parties can tie for the last seat, all we can say a priori is that $k \leq r(a^*) \leq k+n$. We thus make an *ansatz* with $r(\overline{a}) \geq k+n$ and $r(\underline{a}) < k$, and try solve for $\overline{a}$ and $\underline{a}$. While it seems not possible to do this exactly, we can obtain sufficiently tight bounds for nearly-arithmetic divisor sequences to guarantee $|\hat{\mathcal{A}}| = O(n)$.

Remark 2 (Numerical stability): We note that all arithmetic computations in SandwichSelect can safely be implemented with imprecise floating-point arithmetic when rounding conservatively, i.e., rounding towards $-\infty$ for $\underline{a}$ and $\underline{j}$, resp. towards $+\infty$ for $\overline{a}$ and $\overline{j}$. Round-off errors may imply a minor slow-down (by making $\hat{\mathcal{A}}$ slightly larger than necessary), but they do not affect correctness since we use the same value $\underline{j}$ for filling $\hat{\mathcal{A}}$ and for adjusting $\hat{k}$.

Remark 3 (Avoid evaluation of $[\![\cdot]\!]$): Functions δ^{-1} resp. $[\![\cdot]\!]$ might be expensive to evaluate in general. We can replace Step 3.1 by $\underline{j} := \max\{0, \lceil {}^{(v_i\underline{a}-\overline{\beta})}/_{\alpha} \rceil\}$ and $\overline{j} := \lfloor {}^{(v_i\overline{a}-\underline{\beta})}/_{\alpha} \rfloor$. This may make $\hat{\mathcal{A}}$ slightly larger, but our upper bound from Lemma 4 on $|\hat{\mathcal{A}}|$ still applies (cf. Equation (8)). Thus SandwichSelect can run without ever evaluating a rank function or computing an inverse of the divisor sequence. Although this may not be a serious concern for the divisor sequences used in applications, it is unclear whether ChengEppsteinSelect can similarly avoid evaluating rank functions precisely.

Remark 4 (Relation to envy-free stick-division): SandwichSelect is based on a generalization of our solution for the *envy-free stick-division* problem [RW18], a task that arose as a subproblem in a cake-cutting protocol [SHA16]. Given n sticks of lengths $L_1, \ldots, L_n$ and an integer k, the task is to find the longest length ℓ so that we can cut k sticks of length exactly ℓ from the given sticks (without gluing pieces together); this is essentially equivalent to apportionment with $d_j = j+1$.

3.3. Proof of Main Result

Towards proving Theorem 1, we first establish a few intermediate results. We will indeed prove the slightly stronger statement that SandwichSelect correctly computes a^* using $O(n)$ arithmetic operations for *any nearly-arithmetic divisor sequence*, not just signpost sequences. We point out that the running time of SandwichSelect is thus *independent* of k, even when k grows much faster than n. The proofs are elementary, but require care to correctly deal with ties and boundary cases, so we give detailed calculations.

We start by expressing the rank function $r(x)$ in terms of δ^{-1}.

Lemma 2 (Rank via continuation): *The rank function $r(x, \mathcal{A})$ satisfies*

$$r(x, \mathcal{A}) \;=\; \sum_{i=1}^{n} \lfloor \delta^{-1}(v_i \cdot x) \rfloor + 1.$$

Proof: By Equation (3) on page 7, it suffices to show that

$$\left|\{j \in \mathbb{N}_0 \mid {}^{d_j}/_{v_i} \leq x\}\right| \;=\; r(x, A_i) \;=\; \lfloor \delta^{-1}(v_i x) \rfloor + 1$$

for each $i \in \{1, \ldots, n\}$. By Lemma 1, $r(y, \{d_0, d_1, \ldots\}) = \lfloor \delta^{-1}(y) \rfloor + 1$ for all $y \in \mathbb{R}$. Since $x \leq d_j/v_i$ if and only if $y = xv_i \leq d_j$, it follows that $r(x, A_i) = r(xv_i, \{d_0, d_1, \ldots\}) = \lfloor \delta^{-1}(v_i x) \rfloor + 1$. □

Next, we show simple sufficient conditions for bounds $[\underline{a}, \overline{a}]$ to contain our target multiplier a^*.

Lemma 3 (Valid slices): *If $\overline{a}$ and $\underline{a}$ are chosen so that they fulfill*

$$\sum_{i=1}^{n} \delta^{-1}(v_i \cdot \underline{a}) \;\leq\; k - n \qquad \text{and} \qquad \sum_{i=1}^{n} \delta^{-1}(v_i \cdot \overline{a}) \;\geq\; k,$$

then $\underline{a} \leq a^* \leq \overline{a}$.

Proof: As a direct consequence of Lemma 2 together with the fundamental bounds $y - 1 < \lfloor y \rfloor \leq y$ on floors, we find that

$$\sum_{i=1}^{n} \delta^{-1}(v_i \cdot x) \;<\; r(x) \;\leq\; \sum_{i=1}^{n} (\delta^{-1}(v_i \cdot x) + 1) \;=\; n \,+\, \sum_{i=1}^{n} \delta^{-1}(v_i \cdot x) \tag{4}$$

for any x. We now first show that any $a < \underline{a}$ is infeasible. There are two cases: if there is a v_i, such that $v_i a \geq d_0$, we get by strict monotonicity of δ^{-1}

$$r(a) \;\underset{(4)}{\leq}\; n \,+\, \sum_{i=1}^{n} \delta^{-1}(v_i \cdot a) \;<\; n \,+\, \sum_{i=1}^{n} \delta^{-1}(v_i \cdot \underline{a}) \;\leq\; k$$

and a is infeasible. If otherwise $v_i a < d_0$ for all i, a must clearly have rank $r(a) = 0$ as it is smaller than any element $a_{i,j} \in \mathcal{A}$. In both cases we found that $a < \underline{a}$ has rank $r(a) < k$.

It remains to show that $a^* \leq \overline{a}$. By assumption we have

$$r(\overline{a}) \;\underset{(4)}{>}\; \sum_{i=1}^{n} \delta^{-1}(v_i \cdot \overline{a}) \;\geq\; k,$$

so $|\mathcal{A} \cap (-\infty, \overline{a}]| > k$. Hence $a^* = \mathcal{A}_{(k)} \in \mathcal{A} \cap (-\infty, \overline{a}]$ and the claim $a^* \leq \overline{a}$ follows. □

The next lemma shows how to compute explicit bounds for a^* for nearly-arithmetic divisor sequences.

Lemma 4 (Sandwich bounds): *Assume the continuation δ of divisor sequence d fulfills*

$$\alpha x + \underline{\beta} \;\leq\; \delta(x) \;\leq\; \alpha x + \overline{\beta}$$

for all $x \in \mathbb{R}_{\geq 0}$ with $\alpha > 0$, $\underline{\beta} \in [0, \alpha]$ and $\overline{\beta} \geq 0$. Then, the pair $(\underline{a}, \overline{a})$ defined by

$$\underline{a} \;:=\; \max\left\{0, \frac{\alpha k - (\alpha - \underline{\beta}) \cdot n}{V}\right\} \quad \text{and} \quad \overline{a} \;:=\; \frac{\alpha k + \overline{\beta} \cdot n}{V} \tag{5}$$

fulfills the conditions of Lemma 3, that is $\underline{a} \leq a^* \leq \overline{a}$. Moreover,

$$|\mathcal{A} \cap [\underline{a}, \overline{a}]| \;\leq\; 2\left(1 + \frac{\overline{\beta} - \underline{\beta}}{\alpha}\right) \cdot n.$$

Proof: We consider the linear divisor sequence continuations

$$\underline{\delta}(j) = \alpha j + \underline{\beta} \qquad \text{and} \qquad \overline{\delta}(j) = \alpha j + \overline{\beta}$$

for all $j \in \mathbb{R}_{\geq 0}$ and start by noting that the inverses are

$$\underline{\delta}^{-1}(x) = {}^{x}\!/_{\alpha} - {}^{\underline{\beta}}\!/_{\alpha} \qquad \text{and} \qquad \overline{\delta}^{-1}(x) = {}^{x}\!/_{\alpha} - {}^{\overline{\beta}}\!/_{\alpha}$$

for $x \geq \underline{\delta}(0) = \underline{\beta}$ and $x \geq \overline{\delta}(0) = \overline{\beta}$, respectively. For smaller x, we are free to choose the value of the continuation from $[-1, 0)$ (cf. (D4)); noting that ${}^{x}\!/_{\alpha} - {}^{\overline{\beta}}\!/_{\alpha} < 0$ for $x < \overline{\beta}$, a choice that will turn out convenient is

$$\underline{\delta}^{-1}(x) \;:=\; \max\left\{\frac{x}{\alpha} - \frac{\underline{\beta}}{\alpha},\, -1\right\} \quad \text{resp.} \quad \overline{\delta}^{-1}(x) \;:=\; \max\left\{\frac{x}{\alpha} - \frac{\overline{\beta}}{\alpha},\, -1\right\}. \tag{6}$$

We state the following simple property for reference; it follows from $\underline{\delta}(j) \leq \delta(j) \leq \overline{\delta}(j)$ and the definition of the inverses (recall that $\underline{\beta} \leq \alpha$):

$$\frac{x}{\alpha} - \frac{\overline{\beta}}{\alpha} \;\leq\; \overline{\delta}^{-1}(x) \;\leq\; \delta^{-1}(x) \;\leq\; \underline{\delta}^{-1}(x) \;\leq\; \frac{x}{\alpha} - \frac{\underline{\beta}}{\alpha}, \qquad \text{for } x \geq 0. \tag{7}$$

Equipped with these preliminaries, we compute

$$\begin{aligned} & \overline{a} \;=\; \frac{\alpha k + \overline{\beta} n}{V}. \\ \iff\; & k \;=\; \frac{\overline{a} V - \overline{\beta} n}{\alpha} \;=\; \sum_{i=1}^{n}\left(\frac{v_i \cdot \overline{a}}{\alpha} - \frac{\overline{\beta}}{\alpha}\right) \underset{(7)}{\leq} \sum_{i=1}^{n} \delta^{-1}(v_i \cdot \overline{a}), \end{aligned}$$

so $\overline{a}$ satisfies the condition of Lemma 3. Similarly, we find

$$\begin{aligned} & \underline{a} \;\leq\; \frac{\alpha k - (\alpha - \underline{\beta}) n}{V}, \\ \iff\; & k \;\geq\; \frac{\underline{a} V + (\alpha - \underline{\beta}) n}{\alpha} \;=\; n + \sum_{i=1}^{n}\left(\frac{v_i \cdot \underline{a}}{\alpha} - \frac{\underline{\beta}}{\alpha}\right) \\ & \qquad \underset{(7)}{\geq}\; n + \sum_{i=1}^{n} \delta^{-1}(v_i \cdot \underline{a}), \end{aligned}$$

that is $\underline{a}$ also fulfills the conditions of Lemma 3.

For the bound on the number of elements falling between $\underline{a}$ and $\overline{a}$, we compute

$$\begin{aligned} \left|\mathcal{A} \cap [\underline{a}, \overline{a}]\right| \;&=\; \sum_{i=1}^{n} \left|A_i \cap [\underline{a}, \overline{a}]\right| \\ &=\; \sum_{i=1}^{n} \left|\left\{ j \in \mathbb{N}_0 \;\middle|\; \underline{a} \leq \frac{d_j}{v_i} \leq \overline{a} \right\}\right| \end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=1}^{n} \Big| \{ j \in \mathbb{N}_0 \mid v_i \cdot \underline{a} \le d_j \le v_i \cdot \overline{a} \} \Big| \\
&= \sum_{i=1}^{n} \Big| \{ j \in \mathbb{N}_0 \mid \delta^{-1}(v_i \cdot \underline{a}) \le j \le \delta^{-1}(v_i \cdot \overline{a}) \} \Big| \\
&\underset{(7)}{\le} \sum_{i=1}^{n} \Big| \{ j \in \mathbb{N}_0 \mid \overline{\delta}^{-1}(v_i \cdot \underline{a}) \le j \le \underline{\delta}^{-1}(v_i \cdot \overline{a}) \} \Big| \qquad (8) \\
&\le \sum_{i=1}^{n} \Big(\underline{\delta}^{-1}(v_i \cdot \overline{a}) - \overline{\delta}^{-1}(v_i \cdot \underline{a}) + 1 \Big) \\
&\underset{(7)}{\le} \sum_{i=1}^{n} \Big(\frac{v_i \cdot \overline{a} - \underline{\beta}}{\alpha} - \frac{v_i \cdot \underline{a} - \overline{\beta}}{\alpha} + 1 \Big) \\
&= \Big(1 + \frac{\overline{\beta} - \underline{\beta}}{\alpha} \Big) \cdot n \; + \; (\overline{a} - \underline{a}) \cdot \frac{V}{\alpha} \\
&\le \Big(1 + \frac{\overline{\beta} - \underline{\beta}}{\alpha} \Big) \cdot n \; + \; \frac{(\alpha + \overline{\beta} - \underline{\beta}) \cdot n}{V} \cdot \frac{V}{\alpha} \\
&= 2 \Big(1 + \frac{\overline{\beta} - \underline{\beta}}{\alpha} \Big) \cdot n. \qquad \square
\end{aligned}
$$

We are now in the position to prove our main result.

Proof of Theorem 1: We construct the multiset $\hat{\mathcal{A}} \subseteq \mathcal{A}$ as the subsequent union of $A_i \cap [\underline{a}, \overline{a}]$, that is

$$
\begin{aligned}
\hat{\mathcal{A}} &= \biguplus_{i=1}^{n} \Big\{ \frac{d_j}{v_i} \in \mathcal{A} \;\Big|\; \underline{j}(i) \le j \le \overline{j}(i) \Big\} \\
&= \biguplus_{i=1}^{n} \Big\{ \frac{d_j}{v_i} \in \mathcal{A} \;\Big|\; \lceil \delta^{-1}(v_i \cdot \underline{a}) \rceil \le j \le \lfloor \delta^{-1}(v_i \cdot \overline{a}) \rfloor \Big\} \\
&= \biguplus_{i=1}^{n} \Big\{ \frac{d_j}{v_i} \in \mathcal{A} \;\Big|\; v_i \cdot \underline{a} \le d_j \le v_i \cdot \overline{a} \Big\} \\
&= \biguplus_{i=1}^{n} \Big\{ \frac{d_j}{v_i} \in \mathcal{A} \;\Big|\; \underline{a} \le \frac{d_j}{v_i} \le \overline{a} \Big\} \\
&= \mathcal{A} \cap [\underline{a}, \overline{a}].
\end{aligned}
$$

By Lemma 4, we know that $\underline{a} \le a^* \le \overline{a}$ for the bounds computed in Step 1, so we get in particular that $a^* \in \hat{\mathcal{A}}$. It remains to show that we calculate $\hat{k}$ correctly. Clearly, we discard with $(a_{i,0}, \ldots, a_{i,\underline{j}-1})$ exactly $\underline{j}$ elements in Step 3.2, that is $|A_i \cap (-\infty, \underline{a})| = \underline{j}$. Therefore, we compute with

$$
\hat{k} \;=\; k - \sum_{i=1}^{n} |A_i \cap (-\infty, \underline{a})| \;=\; r(a^*, \mathcal{A}) - |\mathcal{A} \cap (-\infty, \underline{a})| \;=\; r(a^*, \hat{\mathcal{A}})
$$

the correct rank of a^* in $\hat{\mathcal{A}}$.

For the running time, we observe that the computations in Step 1 and Step 2 are easily done with $O(n)$ primitive instructions. The loop in Step 3 and therewith Step 3.1 and Step 3.3 are executed n times. The overall number of set operations in Step 3.2 is $|\hat{\mathcal{A}}| = O(n)$ (cf. Lemma 4). Finally, Step 4 runs in time $O(|\hat{\mathcal{A}}|) \subseteq O(n)$ when using a (worst-case) linear-time rank selection algorithm (e. g., the median-of-medians algorithm [Blu+73]). □

4. Comparison of algorithms

We now report from an extensive empirical comparison of all algorithms for divisor methods of apportionment that we found reported in the literature. A more complete discussion of our results is given in our technical report [RW17]; all source codes are available on GitHub [RW15]. For the reader's convenience, we first briefly summarize the algorithms that have not yet been introduced in this article.

4.1. Iterative methods

A naive implementation of the iterative apportionment method (Section 2.3) takes time $\Theta(kn)$; using a priority queue, this can be sped up to $O(k \log n)$.

Pukelsheim [Puk14] notes that the above iterative method can be vastly improved in many instances by starting from a more intelligently chosen initial value for $\mathbf{s}$. His *jump-and-step* algorithm [Puk14, Section 4.6] can be formulated using our notation as follows:

Algorithm 2: JumpAndStep$_d(\mathbf{v}, k)$:

Step 1 Compute an estimate a for a^*; the exact value depends on d:

a) If d is a *stationary signpost sequence*, i.e., $d_j = \alpha j + \beta$ and $\beta/\alpha \in [0, 1]$, then set $a := \frac{\alpha}{V}\big(k + n \cdot (\beta/\alpha - 1/2)\big)$.

b) If d is a *stationary jumppoint sequence*, i.e., $d_j = \alpha j + \beta$ but $\beta/\alpha \notin [0, 1]$ then set $a := \frac{\alpha}{V}\big(k + n \cdot \lfloor \beta/\alpha \rfloor\big)$.

c) Otherwise set $a := \frac{\alpha k}{V}$.

Step 2 Initialize $s_i = \lfloor \delta^{-1}(v_i \cdot a) \rfloor + 1$ for $i = 1, \ldots, n$.

Step 3 While $\sum s_i \neq k$

Step 3.1 If $\sum s_i < k$ set $I := \arg\max_{i=1}^{n} v_i/d_{s_i}$ and $s_I := s_I + 1$;
else set $I := \arg\min_{i=1}^{n} v_i/d_{s_i}$ and $s_I := s_I - 1$.

The performance of JumpAndStep clearly depends on the initial distance from the house size, $\Delta_a := \big(\sum_{i=1}^{n} \lfloor \delta^{-1}(v_i \cdot a) \rfloor + 1\big) - k$: the running time is in $\Theta(n + |\Delta_a| \cdot \log n)$ when using priority queues for Step 3.1. With initial estimate $a = k/V$, we have $\Delta_a \leq n$ for any *signpost sequence* [DK99, Prop. 1], yielding an $O(n \log n)$ method overall. We multiply by α in the formula for a to account for nearly arithmetic sequences that are not signposts. Slight

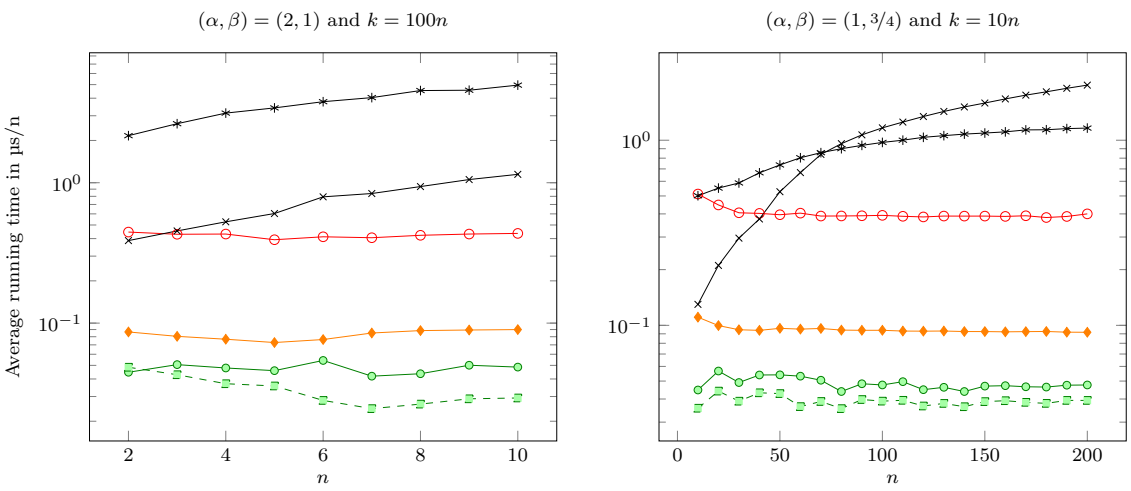

Figure 1: This figure shows average running times on a logarithmic scale for SandwichSelect, ChengEppsteinSelect, JumpAndStep with naive resp. priority-queue minimum selection, and IterativeMethod with naive resp. priority-queue minimum selection, normalized by dividing by the number of parties n. The inputs are random apportionment instances with vote counts v_i drawn i.i.d. uniformly from $[1, 3]$. The numbers of parties n, house size k and method parameters (α, β) have been chosen to resemble national parliaments in Europe (left) and the U.S. House of Representatives (right), respectively.

improvements to $|\Delta_a| \leq n/2$ are possible for *stationary signpost sequences*, $d_j = j + \beta$ with $\beta \in [0, 1]$, using $a = (k+n(\beta-\frac{1}{2}))/V$ [Puk14, Section 6.1]; this corresponds to the case (a) in Step 1. This bound for Δ_a is best possible for the worst case [Puk14, Chap. 6]; it is therefore not possible to obtain a worst-case linear-time algorithm based on JumpAndStep.

4.2. Running Time Comparison

We have implemented all discussed algorithms in Java and conducted a running-time study to compare the practical efficiency of the methods. We use artificial instances; for a given number of parties n, house size k and divisor sequence, we draw multiple vote vectors $\mathbf{v}$ at random according to different distributions. We fix k to a multiple of n and consider arithmetic divisor sequences of the form $(\alpha j + \beta)_{j \in \mathbb{N}_0}$.

We focus on two scenarios here: one resembling current political applications and one exhibiting the worst-case behavior of JumpAndStep; see our technical report [RW17] for a more comprehensive evaluation. We note that in the context of democratic elections, the effort of tallying up the votes likely dwarfs the effort spent for apportioning afterwards; similarly for census data. However, the algorithmic tasks solved in this context are fundamental optimization problems interesting in their own right. We therefore do not want to limit ourselves to the characteristics of specific current applications.

The implementation of ChengEppsteinSelect posed the complication mentioned in Re-

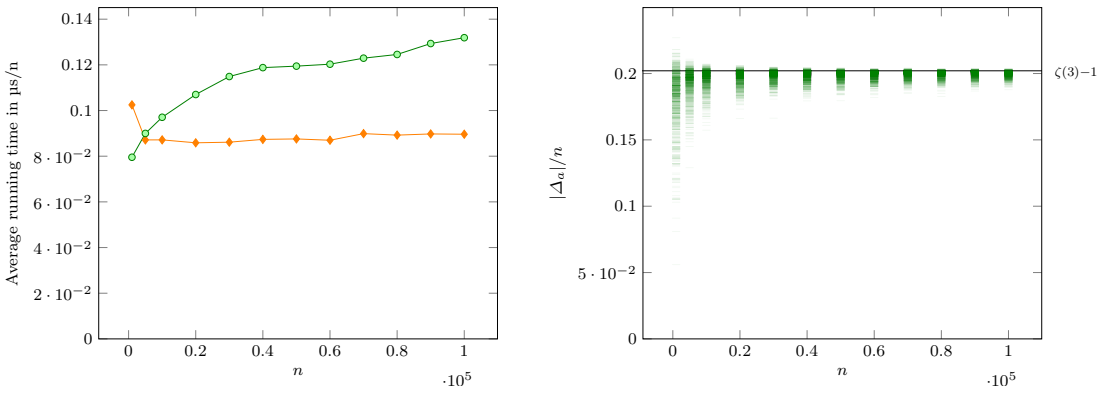

Figure 2: The left plot shows normalized running times of SandwichSelect —♦— and JumpAndStep —○— on instances with $k = 2n$ and Pareto(3)-distributed v_i for $(\alpha, \beta) = (1, 0.001)$. The right plot shows $|\Delta_a|/n$ for all 1000 runs per n; it shows that the expected $|\Delta_a|$ seems to converge to a constant fraction of n in this case. The challenge of this family of apportionment instances lies in the heavy tail of the votes distribution: the majority of parties will get 2 seats unless there are sufficiently many sufficiently popular parties in the instance. Since JumpAndStep's initial estimate only considers V, which for large n is dominated by the vast majority of parties with few votes, the initial seat allocation will give most parties 2 seats and additional seats to the popular parties. More precisely, the expected value of a is $1 + \frac{2}{3}\beta = 1.000\overline{6}$; fixing $a = 1$, the expected number of allocated seats is $(1 + \zeta(3))n \approx 2.2021n$, so the expected allocation in excess of $k = 2n$ is $(\zeta(3) - 1)n$, which matches the data very well.

mark 1. To not unduly slow it down in our running time study, we adopted a fast ad-hoc solution of adding a small constant ϵ before computing floors of floating-point numbers. We could manually determine a suitable ϵ for our benchmark inputs, but we point out that this approach will in general lead to incorrect results (if vote counts are very close).

Figure 1 shows the results of two experiments with practical parameter choices. It is clear that JumpAndStep dominates the field, although SandwichSelect comes close. All other algorithms are substantially slower. As shown in our report [RW17], these observations are stable across many parameter choices. It is worth noting that for small instances, the priority-queue based implementations are slower than the sequential-search based implementations of the iterative method. This is likely due to the initialization overhead for the priority queue.

4.3. Super-linear worst case for JumpAndStep

While JumpAndStep outperforms SandwichSelect for parameters modeling realistic political scenarios – where its initial jump brings it close to the desired house size – in other configurations, it clearly exhibits superlinear behavior; Figure 2 shows such a scenario. Although the sizes are beyond current political applications, for sufficiently large n this makes JumpAndStep slower than SandwichSelect.

Our report [RW17] further shows that SandwichSelect has much smaller variance in running times compared to JumpAndStep, both when varying the individual vote vectors and the used divisor sequences.

In summary, we see that, despite its $\omega(n)$ worst case, JumpAndStep is very fast for many scenarios, and is the best choice for small inputs. SandwichSelect allows for a robust implementation and provides reliable performance across all tested scenarios, independent of divisor sequence and vote distributions; for large instances of the apportionment optimization problem, it is the fastest choice available.

5. Conclusion

We have shown that divisor methods of apportionment can be implemented by a simple and numerically stable algorithm, SandwichSelect, that achieves the optimal linear complexity even in the worst case; the same algorithm works for any rounding rule. The algorithm is simple to state and implement, but its efficiency derives from a close study of the structure of the problem. This concludes the quest for a robust and worst-case efficient implementation of divisor methods.

A closely related area where this quest has not conclusively been achieved is bi-proportional apportionment (double proportionality) [BD89b; BD89a; Puk14]. We leave the question whether new insights from the one-dimensional version can be put to good use in the two-dimensional variant for future work.

Acknowledgments

We thank Chao Xu for pointing us towards the work by Cheng and Eppstein [CE14] and noting that the problem of envy-free stick-division [RW18] is related to proportional apportionment as discussed there. He also observed that our approach for cutting sticks – the core ideas of which turned out to carry over to this article – could be improved to run in linear time.

Furthermore, we owe thanks to an anonymous reviewer whose constructive feedback sparked broad changes which have greatly improved the article over its first incarnation.

Conflict of interest

The authors declare that they have no conflict of interest.

References

[BD89a] M. L. Balinski and G. Demange. "Algorithms for proportional matrices in reals and integers." In: *Mathematical Programming* 45.1-3 (Aug. 1989), pp. 193–210. DOI: 10.1007/bf01589103.

[BD89b] M. L. Balinski and G. Demange. "An Axiomatic Approach to Proportionality Between Matrices." In: *Mathematics of Operations Research* 14.4 (Nov. 1989), pp. 700–719. DOI: 10.1287/moor.14.4.700.

[Blu+73] Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. "Time Bounds for Selection." In: *Journal of Computer and System Sciences* 7.4 (Aug. 1973), pp. 448–461. DOI: 10.1016/S0022-0000(73)80033-9.

[BY01] Michel L. Balinski and H. Peyton Young. *Fair Representation. Meeting the Ideal of One Man, One Vote*. 2nd. Brookings Institution Press, 2001. ISBN: 978-0-8157-0111-8.

[CE14] Zhanpeng Cheng and David Eppstein. "Linear-time Algorithms for Proportional Apportionment." In: *International Symposium on Algorithms and Computation (ISAAC) 2014*. Springer, 2014. DOI: 10.1007/978-3-319-13075-0_46.

[DK99] Gregor Dorfleitner and Thomas Klein. "Rounding with multiplier methods: An efficient algorithm and applications in statistics." In: *Statistical Papers* 40.2 (Apr. 1999), pp. 143–157. DOI: 10.1007/bf02925514.

[FS09] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. (available on author's website: http://algo.inria.fr/flajolet/Publications/book.pdf). Cambridge University Press, 2009. ISBN: 978-0-52-189806-5.

[GKP94] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete mathematics: a foundation for computer science*. Addison-Wesley, 1994. ISBN: 978-0-20-155802-9.

[Puk14] Friedrich Pukelsheim. *Proportional Representation. Apportionment Methods and Their Applications*. 1st ed. Springer, 2014. ISBN: 978-3-319-03855-1. DOI: 10.1007/978-3-319-03856-8.

[RW15] Raphael Reitzig and Sebastian Wild. *Companion Source Code*. revision db43ee7f05. 2015. URL: https://github.com/reitzig/2015_apportionment.

[RW17] Raphael Reitzig and Sebastian Wild. *A Practical and Worst-Case Efficient Algorithm for Divisor Methods of Apportionment*. 2017. arXiv: 1504.06475 [cs.DS].

[RW18] Raphael Reitzig and Sebastian Wild. "Building Fences Straight and High: An Optimal Algorithm for Finding the Maximum Length You Can Cut k Times from Given Sticks." In: *Algorithmica* 80.11 (Nov. 2018), pp. 3365–3396. DOI: 10.1007/s00453-017-0392-3.

[SHA16] Erel Segal-Halevi, Avinatan Hassidim, and Yonatan Aumann. "Waste Makes Haste: Bounded Time Algorithms for Envy-Free Cake Cutting with Free Disposal." In: *ACM Transactions on Algorithms* 13.1 (Dec. 2016), pp. 1–32. DOI: 10.1145/2988232.

[Zac06] Martin Zachariasen. *Algorithmic Aspects of Divisor-Based Biproportional Rounding*. Tech. rep. 06/05. University of Copenhagen, 2006.

A. Notation Index

In this section, we collect the notation used in this paper.

Generic Mathematical Notation

$\lfloor x \rfloor$, $\lceil x \rceil$ . . . . . floor and ceiling functions, as used in [GKP94].

$[\![\cdot]\!]$ . . . . . used rounding rule; see Section 2.2

$\llbracket \cdot \rrbracket$ . . . . . set-valued floor; $\llbracket x \rrbracket = \{\lfloor x \rfloor\}$ if $x \notin \mathbb{N}$ and $\llbracket n \rrbracket = \{n-1, n\}$ for $n \in \mathbb{N}$.

$O(f(n))$, Ω, Θ, $\sim$ . . . asymptotic notation as defined, e.g., in [FS09, Section A.2].

$M_{(k)}$ . . . . . The kth smallest element of (multi)set/vector M (assuming it exists); if the elements of M can be written in non-decreasing order, M is given by $M_{(1)} \leq M_{(2)} \leq M_{(3)} \leq \cdots$.

$\mathbf{x} = (x_1, \ldots, x_d)$ . . . to emphasize that $\mathbf{x}$ is a vector, it is written in **bold**

$\mathcal{M}$ . . . . . to emphasize that $\mathcal{M}$ is a multiset, it is written in calligraphic type.

$\mathcal{M}_1 \uplus \mathcal{M}_2$ . . . . . multiset union; multiplicities add up.

Notation Specific to the Problem

party, seat, vote (count), house size
Parties are assigned seats (in parliament), so that the number of seats s_i that party i is assigned is (roughly) proportional to that party's vote count v_i and the overall number of assigned seats equals the house size k.

$d = (d_j)_{j=0}^{\infty}$ . . . . . the divisor sequence used in the highest averages method; d must be a nonnegative, (strictly) increasing and unbounded sequence.

δ, δ^{-1} . . . . . a continuation of $j \mapsto d_j$ on the reals and its inverse.

n . . . . . number of parties in the input.

$\mathbf{v}$, v_i . . . . . $\mathbf{v} = (v_1, \ldots, v_n) \in \mathbb{Q}_{>0}^n$, vote counts of the parties in the input.

V . . . . . the sum $v_1 + \cdots + v_n$ of all vote counts.

k . . . . . $k \in \mathbb{N}$, the number of seats to be assigned; also called house size.

$\mathbf{s}$, s_i . . . . . $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{N}_0$, the number of seats assigned to the respective parties; the result.

$a_{i,j}$ . . . . . $a_{i,j} := d_j / v_i$, the ratio used to define divisor methods; i is the party, j is the number of seats i has already been assigned.

A_i . . . . . For party i, $A_i := \{a_{i,0}, a_{i,1}, a_{i,2}, \ldots\}$ is the list of (reciprocals of) party i's ratios.

a . . . . . We use a as a free variable when an arbitrary $a_{i,j}$ is meant.

$\mathcal{A}$ . . . . . $\mathcal{A} := A_1 \uplus \cdots \uplus A_n$ is the multiset of all averages.

$r(x, \mathcal{A})$ . . . . . the rank of x in $\mathcal{A}$, that is the number of elements in multiset $\mathcal{A}$ that are no larger than x; $r(x)$ for short if $\mathcal{A}$ is clear from context.

a^* . . . . . the ratio $a^* = a_{i^*,j^*}$ selected for assigning the last, i.e., kth seat.

$\underline{a}$, $\overline{a}$ . . . . . lower and upper bounds on candidates $\underline{a} \leq a \leq \overline{a}$ such that still $a^* \in \mathcal{A} \cap [\underline{a}, \overline{a}]$.