Consensus-Halving: Does It Ever Get Easier?

ARIS FILOS-RATSIKAS, University of Liverpool, United Kingdom
ALEXANDROS HOLLENDER, University of Oxford, United Kingdom
KATERINA SOTIRAKI, Massachusetts Institute of Technology, USA
MANOLIS ZAMPETAKIS, Massachusetts Institute of Technology, USA

In the ε-*Consensus-Halving* problem, a fundamental problem in fair division, there are n agents with valuations over the interval $[0, 1]$, and the goal is to divide the interval into pieces and assign a label "+" or "−" to each piece, such that every agent values the total amount of "+" and the total amount of "−" almost equally. The problem was recently proven by Filos-Ratsikas and Goldberg [18, 19] to be the first "natural" complete problem for the computational class PPA, answering a decade-old open question.

In this paper, we examine the extent to which the problem becomes easy to solve, if one restricts the class of valuation functions. To this end, we provide the following contributions. First, we obtain a strengthening of the PPA-hardness result of [19], to the case when agents have *piecewise uniform* valuations with only *two blocks*. We obtain this result via a new reduction, which is in fact conceptually much simpler than the corresponding one in [19]. Then, we consider the case of *single-block (uniform) valuations* and provide a parameterized polynomial time algorithm for solving ε-*Consensus-Halving* for any ε, as well as a polynomial-time algorithm for $\varepsilon = 1/2$; these are the first algorithmic results for the problem. Finally, an important application of our new techniques is the first hardness result for a generalization of Consensus-Halving, the Consensus-$1/k$-Division problem [33]. In particular, we prove that ε-Consensus-1/3-Division is PPAD-hard.

CCS Concepts: • **Theory of computation → Complexity classes**; **Problems, reductions and completeness**; *Approximation algorithms analysis*.

Additional Key Words and Phrases: consensus-halving; complexity; TFNP; fair division

ACM Reference Format:
Aris Filos-Ratsikas, Alexandros Hollender, Katerina Sotiraki, and Manolis Zampetakis. 2020. Consensus-Halving: Does It Ever Get Easier?. In *Proceedings of the 21st ACM Conference on Economics and Computation (EC '20), July 13–17, 2020, Virtual Event, Hungary*. ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3391403.3399527

1 INTRODUCTION

The topic of *fair division* has been in the focus of research in economics and mathematics, since the late 1940s and the pioneering works of Banach, Knaster and Steinhaus [34], who developed the associated theory. The related literature contains many interesting problems, with the most celebrated perhaps being the problems of *envy-free cake-cutting* and *equitable cake-cutting*, for which a plethora of results have been obtained. More recently, the computer science literature has made a significant contribution in studying the computational complexity of these problems, and attempting to design efficient algorithms for several of their variants [4–6, 15].

EC '20, July 13–17, 2020, Virtual Event, Hungary

ACM ISBN 978-1-4503-7975-5/20/07…$15.00
https://doi.org/10.1145/3391403.3399527

Another classical problem in fair division, whose study originates back to as early as the 1940s and the work of Neyman [30], is the *Consensus-Halving problem*[1] [33]. In this problem, there is a set of n agents with valuation functions over the $I = [0, 1]$ interval. The goal is to divide the interval into pieces using at most n cuts, and assign a label from $\{+, -\}$ to each piece, such that every agent values the total amount of I labeled "+" and the total amount of I labeled "−" equally. Similarly to other well-known problems in fair division, the existence of a solution to the Consensus-Halving problem is *always* guaranteed, and can be proven via the application of a fixed-point theorem; here the *Borsuk-Ulam theorem* [8]. As a matter of fact, the problem is a continuous analogue of the well-known Necklace Splitting problem [2, 22], whose existence of a solution is typically established via an existence proof for the continuous version.

The Consensus-Halving problem attracted attention in the literature of computer science recently, due to the breakthrough results of Filos-Ratsikas and Goldberg [18, 19] who studied the computational complexity of the approximate version, in which there is a small allowable discrepancy ε between the values of the two portions. First, in [18], the authors proved that ε-Consensus-Halving for inverse-exponential ε is complete for the computational class PPA, defined by Papadimitriou [32]. This was the first PPA-completeness result for a "natural" problem, i.e., a computational problem that does not have a polynomial-sized circuit explicitly in its definition, answering an open question from Papadimitriou [32], reiterated multiple times over the years [1, 25]. Then in [19], the authors strengthened their hardness result to the case of inverse-polynomial ε, which also established the PPA-completeness of the Necklace Splitting problem for 2 thieves.

Despite the aforementioned results, the complexity of the problem is not yet well understood. Does the problem remain hard if one restricts attention to classes of simple valuation functions? Note that the reduction of [18, 19] uses instances with *piecewise constant* valuation functions with *polynomially many* pieces. On the opposite side, are there efficient algorithms for solving special cases of the problem? What if we allow a larger number of cuts?

1.1 Our Results

Towards understanding the complexity of Consensus-Halving, we present the following results.

- We prove that ε-Consensus-Halving is PPA-complete, even when the agents have *two-block uniform* valuations, i.e., valuation functions which are *piecewise uniform* over the interval and assign non-zero value on at most *two* pieces. This result holds even when ε is inverse-polynomial, and extends to the case where the number of allowable cuts is $n + n^{1-\delta}$, for some constant $\delta > 0$.

  This is an important strengthening of the results in [18, 19] which require the agents to have piecewise constant valuations with polynomially many non-uniform blocks. En route to this result, we obtain a significant simplification to the proof of [18, 19], which uses new gadgets for the encoding of the circuit of high-dimensional Tucker (see Definition 2.4), which we reduce from. Our new reduction also gives a simplified proof of PPA-completeness for Necklace Splitting with 2 thieves [18, 19].

- We study the case of *single-block valuations* and provide the first algorithmic results for the problem.[2] Specifically, we present:

[1] The name "Consensus-Halving" is attributed to Simmons and Su [33], although the problem has been studied under different names in the past. For example, it is also known as *The Hobby-Rice theorem* [26], or *continuous necklace splitting* [2].
[2] To be precise, we provide the first such results for the version of the problem with n agents and n cuts. For a large number of cuts, Brams and Taylor [9] present algorithms for ε-approximate solutions. Crucially, these algorithms require a number of cuts which grows as ε decreases, while our results for more than n cuts are not dependent on ε.

- an algorithm for any ε, whose running time is polynomial in $1/\varepsilon$ and a parameter d related to the maximum number of overlapping blocks.
- a polynomial-time algorithm for 1/2-Consensus-Halving.

We complement our main results with a simple algorithm based on linear programming, which solves the problem for single-block valuations in polynomial-time, if one is allowed to use $2n - \ell$ cuts, for any constant ℓ.

- As an application of the new ideas developed in our reduction, we obtain the first hardness result for a generalization of ε-Consensus-Halving, known as ε-*Consensus*-$1/k$-*Division*, for $k \geq 3$. Specifically, we prove that ε-*Consensus*-1/3-*Division* is PPAD-hard, when ε is inverse-exponential.

Several of our proof are omitted due to lack of space, but can be found in the full version of the paper.

1.2 Discussion and Related Work

The study of the Consensus-Halving problem originates back to the early 1940s and the work of Neyman [30]. The first proof of existence for n cuts can be traced back to the 1965 theorem of Hobby and Rice [26]. The problem was famously studied in the context of Necklace Splitting, being a continuous analogue of the latter problem; in fact, most known proofs for Necklace Splitting go via the continuous version[3] [3, 22]. The name *Consensus-Halving* is attributed to Simmons and Su [33], who studied the continuous problem independently, and came up with a constructive proof of existence. Their construction, although yielding an exponential-time algorithm, was later adapted by Filos-Ratsikas et al. [17] to prove that the problem lies in the computational class PPA.

The class PPA was defined by Papadimitriou [32] in his seminal paper in 1994, in which he also defined several other important subclasses of TFNP [28], the class of *Total Search Problems in NP*, i.e., problems that *always* have solutions which are efficiently verifiable. Among those classes, the class PPAD has been very successful in capturing the complexity of many interesting computational problems [11, 21, 23, 29], highlighted by the celebrated result of Daskalakis et al. [12] and Chen et al. [10] about the PPAD-completeness of computing a Nash equilibrium. On the contrary, since the definition of the class, PPA was not known to contain any natural complete problems, but rather mostly versions of PPAD-complete problems of a topological nature, defined on non-orientable spaces [14, 25]. In 2015, Aisenberg et al. [1] showed that the computational version of Tucker's Lemma [35], already shown to be in PPA by Papadimitriou [32], is actually complete for the class.

Using the latter result as a starting point, Filos-Ratsikas and Goldberg [18] proved that ε-Consensus-Halving is PPA-complete when ε is inverse exponential. This was a breakthrough result in the following sense: it was the first PPA-completeness result for a "natural" computational problem, where the term "natural" takes the specific meaning of a problem that does not have a polynomial-sized circuit in its definition. The quest for such problems that would be complete for PPA was initiated by Papadimitriou himself [32] and was later brought up again by several authors, including Grigni [25] and Aisenberg et al. [1]. In the same paper, the authors also provided a computational equivalence between the ε-Consensus-Halving problem and the well-known Necklace Splitting problem of Alon [2] for 2 thieves [3, 22], when ε is inverse-polynomial. In [19], the authors strengthened their result to ε being inverse-polynomial, which, together with the aforementioned result from [18], also provided a proof for the PPA-completeness of Necklace Splitting. As we mentioned earlier, besides being a strengthening, our PPA-hardness proof for ε-Consensus-Halving is a notable simplification over that of [19], and importantly, it holds for ε

[3] This is true for the case of 2 thieves. For k thieves, the proofs go via the Consensus-$1/k$-Division problem instead.

which is inverse-polynomial. Therefore, we also obtain a new, simplified proof of PPA-hardness for Necklace Splitting with 2 thieves.

For constant ε, the only hardness result that we know is the PPAD-hardness of Filos-Ratsikas et al. [17], who also show that when $n-1$ cuts are allowed, deciding whether a solution exists is NP-hard. Recently, Deligkas et al. [13] studied the complexity of *exact* Consensus-Halving and showed that the problem is FIXP-hard. Interestingly, the authors also introduced a new computational class, called BU (for Borsuk-Ulam) and showed that the problem lies in that class, leaving open the question of whether it is BU-complete.

If we generalize the number of labels to $\{1, 2, \ldots, k\}$ rather that $\{+, -\}$, and we allow $(k-1)n$ cuts rather than only n, then we obtain a generalization of the Consensus-Halving problem which was referred to as *Consensus-1/k-Division* in [33]. The existence of a solution for this problem can be proved via fixed-point theorems that generalize the Borsuk-Ulam theorem [2, 7], however very little is known about its complexity. One might feel inclined to believe that Consensus-1/k-Division is a harder problem that Consensus-Halving; however, note that in the former problem, we have more cuts at our disposal. In fact, Filos-Ratsikas and Goldberg [19] conjectured that the complexities of the problems for different values of k are incomparable, and are characterized by different complexity classes. The complexity classes that are believed to be the most related are called PPA-k, defined also by Papadimitriou [32] in his original paper; we refer the reader to the recent papers of [24, 27] for a more detailed discussion of these classes.

Before our paper, virtually nothing was known about the hardness of the problem when $k \geq 3$. While the techniques in [19] were highly reliant on the presence of only two labels, our ideas do carry over to the case when $k = 3$, which enables us to prove our PPAD-hardness result. While we do not expect the problem for $k \geq 3$ to be PPAD-complete, our proof offers important intuition about the intricacies of the problem and could be useful for proving stronger hardness results in the future.

2 PRELIMINARIES

We start with the definition of the ε-approximate version of the Consensus-Halving problem.

Definition 2.1 (ε-Consensus-Halving). Let $k \geq 2$. We are given $\varepsilon > 0$ and a set C of continuous probability measures $\mu_1, \ldots, \mu_n$ on $I = [0, 1]$. The probability measures are given by their density functions on I. The goal is to partition the unit interval into 2 (not necessarily connected) pieces I^+ and I^- using at most n cuts, such that $|\mu_j(I^+) - \mu_j(I^-)| \leq \varepsilon$ for all agents $j \in \{1, \ldots, n\}$.

We will refer to the probability measures $\mu_1, \ldots, \mu_n$ as *valuation functions* or simply *valuations*. While the existence and PPA-membership results hold more generally, in this paper, we will restrict our attention to the case when the valuation functions are *piecewise constant*. These can be represented explicitly in the input as endpoints and heights of value blocks.

Definition 2.2 (Piecewise constant valuation functions). A valuation function μ_i is *piecewise constant* over an interval I, if the domain can be partitioned into a finite set of intervals such that the density of μ_i is constant over each interval.

Piecewise constant functions are often referred to as *step functions*.

Definition 2.3 (Uniform valuation functions). We will consider the following subclasses of piecewise constant valuation functions.

- ***Piecewise Uniform:***, The domain can be partitioned into a finite set of intervals such that the density of μ_i is either v_i or 0 over each interval, for some constant v_i.

- d-***block Uniform:*** The domain can be partitioned into a finite set of intervals, such that in at most d of those the density of μ_i is v_i and everywhere else it is 0, for some constant v_i.
- 2-***block Uniform:*** d-block uniform valuations for $d = 2$.
- ***Single-block:*** d-block uniform valuations for $d = 1$. Here we omit the term "uniform", as there is only a single value block.

Obviously, piecewise constant $\supseteq$ piecewise uniform $\supseteq$ 2-block uniform $\supseteq$ single-block.

2.1 The Computational Classes PPA and PPAD

As we mentioned in the introduction, Consensus-Halving is a *Total Search Problem in NP*, i.e., a problem with a guaranteed solution which is verifiable in polynomial time. The corresponding class is the class TFNP [28]. Formally, a binary relation $P(x, y)$ is in the class TFNP if for every x, there exists a y of size bounded by a polynomial in $|x|$ such that $P(x, y)$ holds and $P(x, y)$ can be verified in polynomial time. The problem is given x, to find such a y in polynomial time.

The subclasses of TFNP that will be relevant for this paper are PPAD and PPA. Intuitively, PPAD is defined with respect to a directed graph of exponential size, which is given *implicitly* as input, via the use of predecessor and successor circuits. PPAD is a subclass of PPA, which is defined similarly, but with respect to an undirected graph and a circuit that outputs the neighbours of a vertex. Formally, these classes are defined via their canonical problems, End-of-Line and Leaf [32], and membership and hardness are established via polynomial-time reductions to and from these problems respectively.

2.2 High-dimensional Tucker

Our reduction in Section 3 will start from the following problem, which is an N-dimensional variant of the $2D$-Tucker problem [1, 32].

Definition 2.4 (high-D-Tucker). An instance of high-D-Tucker consists of a labeling $\lambda : [8]^N \to \{\pm 1, \ldots, \pm N\}$ computed by a Boolean circuit. We further assume that the labeling is antipodally anti-symmetric (i.e. for all x on the boundary of $[8]^N$ it holds that $\lambda(\overline{x}) = -\lambda(x)$ where $\overline{x}_i = 9 - x_i$ for all i), which can be enforced syntactically. A solution consists of two points $x, y \in [8]^N$ with $\lambda(x) = -\lambda(y)$ and $\|x - y\|_\infty \leq 1$.

Filos-Ratsikas and Goldberg [19] showed that the problem is PPA-hard, when the domain is $[7]^N$ instead of $[8]^N$. We adapt the hardness to the case of Definition 2.4 in the theorem below.

Theorem 2.5. *high-D-Tucker is* PPA*-complete.*

3 CONSENSUS-HALVING WITH TWO-BLOCK UNIFORM VALUATIONS IS PPA-HARD

In this section, we present our first result, regarding the PPA-hardness of Consensus-Halving.

Theorem 3.1. *ε-Consensus-Halving is PPA-hard, when ε is inverse-polynomial and the agents have two-block uniform valuations.*

As we mentioned in the Introduction, Theorem 3.1 is a strengthening of the result of [19], which requires the valuation functions to have a polynomial number of value blocks, and which is seemingly very difficult to extend to two-block uniform valuations. To achieve this stronger result, we have to develop new gadgetry, based on a new interpretation of the cut positions with respect to the positions of points in the domain of high-D-Tucker. As it turns out, this new interpretation allows us to obtain a new proof of the main theorem of [19], one which is conceptually much simpler, even though it actually applies to more restricted valuations.

Before we proceed, we first remark the following. In [17] (where the PPAD-hardness of ε-Consensus-Halving was proven for constant ε) the authors presented a simple argument that allowed them to extend their hardness result to $n + c$ cuts, where c is some constant. The idea is to make $c + 1$ *completely disjoint* copies of the instance of ε-Consensus-Halving, and solve it using $n + c$ cuts. One of the copies would have to be solved using at most n cuts, which is a PPAD-hard problem. We observe that the same principle applies generically (beyond PPAD-hardness and also to the results of [18, 19]), and in fact extends to $n + n^{1-\delta}$ cuts, where $\delta > 0$ is some constant. From Theorem 3.1, we obtain the following corollary.

Corollary 3.2. *ε-Consensus-Halving is PPA-hard, when ε is inverse-polynomial and the agents have two-block uniform valuations, even when one is allowed to use $n + n^{1-\delta}$ cuts, for constant $\delta > 0$.*

We are now ready to prove Theorem 3.1. We first provide an overview of the reduction and we highlight the main simplifications over the proof of [19]. Then we proceed to formally present the proof of Theorem 3.1.

3.1 Overview of the reduction

We are given an instance of high-D-Tucker, namely a labeling $\lambda : [8]^N \to \{\pm 1, \ldots, \pm N\}$ computed by a Boolean circuit. We will show how to construct an instance of Consensus-Halving in polynomial time such that any ε-approximate solution yields a solution to the high-D-Tucker instance (for some inversely-polynomial ε). The complexity will be measured with respect to the representation size of the high-D-Tucker instance, i.e., the size of the circuit λ (which is also at least N).

For clarity and convenience, the instance of Consensus-Halving we will construct will not be defined on the domain $[0, 1]$, but instead on some interval $[0, M]$, where M is bounded by a polynomial in the size of the high-D-Tucker circuit λ. It is easy to transform this into an instance on $[0, 1]$ by just re-scaling the valuation functions, namely scaling down the positions of the blocks by M and scaling up the heights of the blocks by M.

Overview. Let us first provide a very high-level description of the instance we construct. Similarly to [19], the left-most end of the instance will be the *Coordinate-Encoding* region. In any solution S to the instance, the way in which this region is divided amongst the labels + and − will represent a point $x \in [-1, 1]^N$. A *circuit-simulator* C will read-in the coordinates of x, perform some computations (including a simulation of λ) and output N values $[C(x)]_1, \ldots, [C(x)]_N \in [-1, 1]$. The circuit-simulator will consist of a set of agents and each agent will implement one gate/operation of the circuit. Unfortunately, the circuit-simulator can sometimes fail to perform the desired computation, so instead of one circuit-simulator C we will actually have a polynomial number $p(N)$ of circuit-simulators $C_1, \ldots, C_{p(N)}$. Each of these circuit-simulators will be performing (almost) the same computation. Finally, we will introduce a *Feedback region* where N feedback agents $f_1, \ldots, f_N$ will implement the *feedback mechanism*. For each $i \in \{1, \ldots, N\}$, feedback agent f_i will ensure that $\frac{1}{p(N)} \sum_{j=1}^{p(N)} [C_j(x)]_i \approx 0$. Namely, it will ensure that the average of the outputs in dimension i is close to zero. We will show that from any solution S to the Consensus-Halving instance, we obtain a solution to the original high-D-Tucker instance.

Encoding of a value in $[-1, 1]$**.** Given any solution S of our instance, every interval I of length 1 of the domain encodes a value in $[-1, 1]$ as follows. Let I^+ and I^- denote the subsets of I labeled respectively + and − in the solution S. Then the value encoded by I, $v_S(I)$, is given by $\mu(I^+) - \mu(I^-)$, where μ is the Lebesgue measure on $\mathbb{R}$. Since there are at most n cuts (where n is the number of agents in the instance), I^+ is the union of at most $n + 1$ disjoint sub-intervals of I and $\mu(I^+)$ is simply the sum of the lengths of these intervals (and the same holds for I^-). It is easy to see that $v_S(I) = 0$

corresponds to I being perfectly shared between + and − in S, whereas $v_S(I) = +1$ corresponds to the whole interval I being labeled +. We will drop the subscript S and just use $v(I)$ in the remainder of this exposition.

Coordinate-Encoding region. The sub-interval $[0, N]$ of the domain is called the *Coordinate-Encoding* region. Indeed, the way in which this region is subdivided amongst the + and − labels in a solution S will encode the coordinates of a point in $x \in [-1, 1]^N$. In more detail, $x_1 \in [-1, 1]$ will be given by $v([0, 1])$, i.e., the value encoded by interval $[0, 1]$. Similarly, $x_2 \in [-1, 1]$ will be given by $v([1, 2])$, $x_3 \in [-1, 1]$ by $v([2, 3])$, etc.

Constant-Creation region. The sub-interval $[N, N+p(N)]$ of the domain is called the *Constant-Creation* region. This region will be used to create the constants that the circuit-simulators need. The circuit-simulator C_1 will read-in the value $v([N, N + 1]) =: \text{const}_1$ and will assume that it corresponds to the value +1. Note that given the constant +1, the circuit-simulator can create any constant $\zeta \in [-1, 1]$ by using a $\times\zeta$-gate (multiplication by the constant ζ). Similarly, the circuit-simulator C_2 will read-in the value $v([N + 1, N + 2]) =: \text{const}_2$ and use it as the constant +1, and so on for $C_3, C_4, \ldots, C_{p(N)}$.

If S is a solution such that the Constant-Creation region does not contain any cut, then the whole region will have the same label, and without loss of generality we can assume that this label is +. Thus, in such a solution S, all the circuit-simulators will indeed read-in the constant +1 from the Constant-Creation region, i.e., we will indeed have $\text{const}_j = +1$ for all $j = 1, \ldots, p(N)$.

Circuit-Simulation regions. For each $j \in \{1, 2, \ldots, p(N)\}$, the sub-interval $[N + p(N) + (j - 1)q, N + p(N) + jq]$ of the domain will be used by the circuit-simulator C_j. The length q used by every circuit-simulator will be upper-bounded by some polynomial in N and the size of the circuit λ. Every circuit-simulator C_j will read-in the coordinates $x_1, \ldots, x_N \in [-1, 1]$ of the point x from the Coordinate-Encoding region, as well as the value $\text{const}_j \in [-1, 1]$ from the Constant-Creation region (and assume that it corresponds to the constant +1). Using these values, C_j will perform some computations, including a simulation of the Boolean circuit λ, and finally output N values $[C_j(x, \text{const}_j)]_1, \ldots, [C_j(x, \text{const}_j)]_N \in [-1, 1]$ into the *Feedback region*.

Feedback region. The Feedback region is located at the right end of the domain and is subdivided into N intervals $F_1, \ldots, F_N$ of length $p(N)$ each. For every $j \in [p(N)]$, let $F_i(j)$ denote the jth sub-interval of length 1 of F_i. The ith output of circuit-simulator C_j will be located in sub-interval $F_i(j)$. In other words, $v(F_i(j)) = [C_j(x, \text{const}_j)]_i$.

Every interval F_i will have a corresponding *feedback agent* f_i, who will ensure that the average of all the outputs in interval F_i is close to zero. In more detail, agent f_i will have a single block of value that covers interval F_i. As a result, this agent will be satisfied only if $\frac{1}{p(N)} \sum_{j=1}^{p(N)} v(F_i(j)) \in [-\varepsilon, \varepsilon]$.

Stray Cuts. Any agent belonging to a circuit-simulator performs a gate-operation. In Section 3.3, we introduce the different types of gates and how they are implemented by agents. One important feature of the agents implementing the gates is that every such agent ensures that at least one cut must lie in a specific interval J of the domain (in any solution S). By construction, we will make sure that these intervals are pairwise disjoint for different agents. Thus, every agent introduced as part of a circuit-simulator will force one cut to lie in a specific interval.

The only agents that are not part of a circuit-simulator are the feedback agents $f_1, \ldots, f_N$. Since the number of cuts in any solution is at most the number of agents, there are at most N cuts that are not constrained to lie in some specific interval. We call these the *free cuts*. The free cuts can theoretically "go" anywhere in the domain and interfere with the correct functioning of the circuit-simulators or the Constant-Creation region. The expected behavior of these N free cuts

is that they should lie in the Coordinate-Encoding region. As such, any of the free cuts that lies outside the Coordinate-Encoding region will be called a *stray cut* (following [19]).

Observation 1. *If there is at least one stray cut, then the point $x \in [-1,1]^N$ encoded by the Coordinate-Encoding region lies on the boundary of $[-1,1]^N$ (i.e., there exists i such that $|x_i| = 1$).*

Stray Cut interference. There are two ways for a stray cut to cause trouble:

(1) it can *corrupt* a circuit, i.e., interfere with the correct functioning of the gates of a circuit-simulator. If the cut lies in the region of circuit-simulator C_j, then it can make a gate output the wrong result (i.e., not perform the desired operation). If the cut lies in the Constant-Creation region and intersects the interval that is used by circuit-simulator C_i to read-in the constant const_j, then it can have an effect such that $|\text{const}_j| \neq 1$. However, in any case, a single stray cut can only interfere with one circuit-simulator in this way. Thus, at most N circuit-simulators can suffer from this kind of interference. We will choose $p(N)$ large enough so that these corrupted circuit-simulators have a very limited influence.
(2) it can interfere with the sign of const_j for many circuit-simulators C_j. Indeed, even a single stray cut can ensure that half of our circuit-simulators read-in the constant $+1$ and the other half read-in the constant -1. We will show that this is actually not a problem, and that it does not produce bogus solutions. Since stray cuts can only occur when x lies on the boundary of $[-1,1]^N$ (Observation 1), the Tucker boundary conditions will be important for this.

Stray cuts that end up in the Feedback region do not have any effect. Indeed, the feedback agents $f_1, \ldots, f_N$ are immune to stray cuts. They always ensure that the average of the outputs is close to zero. Thus, a stray cut can only influence the outputs that a feedback agent sees (as detailed above), but not its functionality.

Circuit-Simulator failure. There are two ways in which a circuit-simulator can fail to have the desired output:

(1) it is corrupted by a stray cut. This can happen to at most N circuit-simulators.
(2) it can fail in extracting the binary bits from (a point close to) x. We will ensure that this can happen to at most N circuit-simulators.

Thus, at most $2N$ circuit-simulators fail, i.e., at least $p(N) - 2N$ circuit-simulators have the desired output.

3.2 Major simplifications compared to [19]

A much cleaner domain. The PPA-hardness of high-D-Tucker was already established in [19] and our version can be obtained from that one using minor modifications, see Theorem 2.5. The corresponding result of [19] is a standard application of the "snake-embedding" technique developed in [10]. However, the reduction in [19] requires (a) a further constraint on how the domain is colored and more importantly (b) the embedding of the high-D-Tucker instance into a Möbius-type simplex domain, in which two facets have been "identified" with each other - one can envision a high-dimensional Möbius strip with an instance of high-D-Tucker in its center, embedding in a high-dimensional simplex. A key step in the reduction is the extension of the labeling of high-D-Tucker to the remainder of the domain, in a way that does not introduce any artificial solutions, and such that solutions to high-D-Tucker can be traced back from solutions on other points on the domain. For this purpose, the authors of [19] develop a rather complicated coordinate transformation, applied to the inputs read from the positions of the cuts. They establish how to compute the transformation and its inverse in polynomial time and how distances in the two coordinate systems (before and after the transformation) are polynomially related. In contrast,

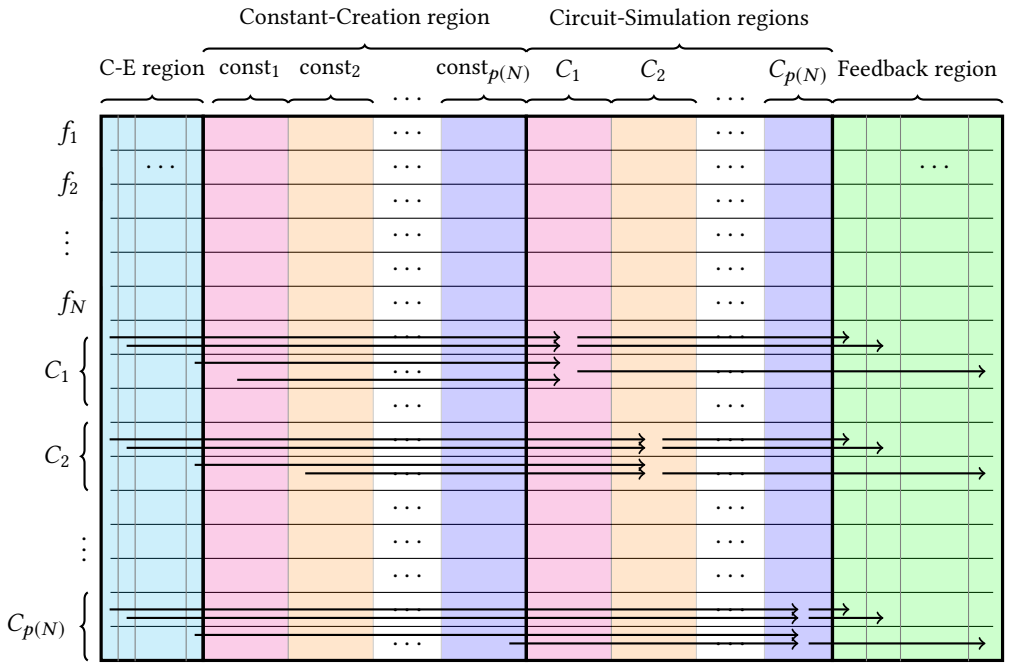

Fig. 1. An overview of the different regions defined in the reduction. The regions corresponding to different Circuit-Simulators are color-coded. An arrow indicates that the region where it is pointing receives inputs from the region from where it is originating. On the left, the different types of agents are shown, namely the feedback agents, as well as the agents corresponding to the different Circuit-Simulators. The Coordinate-Encoding region and the Feedback Region are divided into sub-intervals, indicated by vertical gray lines, as detailed in Section 3.1.

our reduction works with the rather clean domain of HIGH-D-TUCKER, avoiding all the unnecessary technical clutter of the domain used in [19].

Simpler gadgetry. Another complication of the proof in [19] is the use of blanket-sensor agents, which constrain the positions of the cuts in the coordinate-encoding region, to ensure that solutions to ε-CONSENSUS-HALVING do not encode points that lie too far from a specific region in the "middle" of the domain, called the "significant region"; this is achieved via appropriate feedback provided by these agents to the coordinate-encoding agents. To make sure that the blanket-sensor agents do not "cancel" each other, extra care must be taken on how the feedback of these agents is designed, giving rise to a series of technical lemmas. Our reduction does not need to use any such agents and is therefore significantly simpler in that regard as well.

Label sequence robustness. The reduction in [19] requires knowledge of the label sequence, i.e., whether the first cut that occurs in the c-e region has the label + or − on its left side. This is fundamental for the design of the gates, as they read the inputs as the distances from the left endpoints of the corresponding designated intervals, unlike our interpretation, which measures the difference between the value of the two labels. Thus, for the disorientation of the domain and to deal with sign flips that happen due to the stray cuts, the authors of [19] employ a pre-processing circuit that uses the first coordinate-detecting agent as a reference agent, when performing computations.

This is again not needed in our case; our equivariant gates ensure that even when the corresponding point lies on the boundary of the HIGH-D-TUCKER domain, the output is computed correctly in a much simpler way.

3.3 Arithmetic Gates

In this section we show how to construct gates which perform various operations on numbers in $[-1, 1]$ with error at most $g(\varepsilon) = 16\varepsilon$, where ε is the error we allow in a CONSENSUS-HALVING solution. Some of these gates will be immune to "corruption" by a stray cut, while others might get corrupted and not work properly.

Recall that in any solution S, any unit length interval I of the domain represents value $v(I) \in [-1, 1]$. We will now show how to perform computations with these values. We let $T : \mathbb{R} \to [-1, 1]$, $z \mapsto \max(-1, \min(1, z))$, i.e., $T[z]$ is the *truncation* of $z \in \mathbb{R}$ in $[-1, 1]$. We will also abuse notation and use $T[x] = (T[x_1], \ldots, T[x_N])$ for $x \in [-1, 1]^N$. At this stage, we assume that ε is sufficiently small for the gates to work (namely $\varepsilon \leq 2^{-10}$ is enough).

We will design *basic gates*, namely ***Multiplication by*** -1 $[G_{\times(-1)}]$, ***Constant*** $\zeta \in [-1, 1] \cap \mathbb{Q}$ $[G_{\zeta}]$ and ***Addition*** $[G_+]$, and *additional gates*, namely ***Copy*** $[G_{\mathbf{copy}}]$, ***Multiplication by*** $k \in \mathbb{N}$ $[G_{\times k}]$ and Boolean Gates, ***Negation*** $[G_\neg]$, ***AND*** $[G_\wedge]$ and ***OR*** $[G_\vee]$. We refer the reader to the full version for the detailed design of the gates.

Remark 1 (Equivariant Gates). Note that the operation performed by any gate is *equivariant*. Namely, if we flip the sign of all inputs, the same output is still valid, but with a flipped sign. For $G_{\times -1}$ and G_+ gates this is obvious. For G_ζ, we have to recall that const_j is the input to the gate. With this interpretation, the equivariance is once again obvious. For the $G_\wedge$-gate the equivariance is a bit more subtle. Note that it uses a $G_{-1/2}$-gate, which uses const_j. Thus, in this case again, if we flip the sign of the inputs b_1, b_2 and const_j, the sign of the output is flipped too.

This property of the gates is not a coincidence. It follows from the way we are encoding values. Note that in any solution S, if we swap the labels $+$ and $-$, the solution remains valid. The only thing that has changed is that for any gate, the sign of all inputs and outputs has been flipped.

It follows that any circuit that we construct out of these gates will be equivariant. Namely, the computation will still be valid if we flip the sign of all inputs (including const_j) and all outputs.

3.4 Circuit-Simulators

In this section we describe the functionality of the circuit-simulators and what it achieves. Recall that every circuit-simulator C_j reads-in inputs $x_1, \ldots, x_N \in [-1, 1]$ from the Coordinate-Encoding region and $\text{const}_j \in [-1, 1]$ from the Constant-Creation region. The circuit-simulator then performs some computations and outputs $[C_j(x, \text{const}_j)]_1, \ldots, [C_j(x, \text{const}_j)]_N \in [-1, 1]$ into the Feedback region. If the circuit-simulator C_j is corrupted (i.e., one of the stray cuts interferes with it), then we will not claim anything about the outputs of C_j. In that case, we will only use the fact that all the outputs must lie in $[-1, 1]$ (which is guaranteed by the way values are represented).

If the circuit-simulator C_j is not corrupted, then we know that all gates will perform correct computations, and we also know that $\text{const}_j \in \{-1, +1\}$. In our construction of C_j, we will be assuming that $\text{const}_j = +1$. However, we will show later that even if $\text{const}_j = -1$, C_j will output something useful.

Phase 1: equi-angle displacement. In the first phase, C_j applies a small displacement to its input x. Namely, for every $i \in [N]$, C_j computes $\widehat{x}_i \approx T[x_i + j\alpha]$, where $\alpha = \frac{1}{16p(N)}$. This is achieved by using a $G_{j\alpha}$-gate to create the constant $j\alpha$ (by using const_j), followed by a G_+-gate to perform the addition $x_i + j\alpha$. However, since $\varepsilon > 0$, the gates might make some error in the computations. Nevertheless, by construction of the gates, we immediately obtain:

Claim 1. *Assume that the circuit-simulator C_j is not corrupted and* $\text{const}_j = +1$. *Then the equi-angle displacement phase outputs $\widehat{x}_i = T[x_i + j\alpha \pm 2g(\varepsilon)]$ for all i.*

Phase 2: bit extraction. In the second phase, C_j extracts the three most significant bits from each $\widehat{x}_i \in [-1, 1]$. These three bits tell us where $\widehat{x}_i$ lies in $[-1, 1]$, namely in which of the eight possible standard intervals of length $1/4$: $[-1, -3/4]$, $[-3/4, -1/2]$, $[-1/2, -1/4]$, $[-1/4, 0]$, $[0, 1/4]$, $[1/4, 1/2]$, $[1/2, 3/4]$, $[3/4, 1]$. Instead of the usual $\{0, 1\}$, our bits will take values in $\{-1, +1\}$. The first bit $b_1 \in \{-1, +1\}$ indicates whether $\widehat{x}_i$ is positive or negative. If $b_1 = +1$, then $\widehat{x}_i \in [0, 1]$. If $b_1 = -1$, then $\widehat{x}_i \in [-1, 0]$. The second bit $b_2 \in \{-1, +1\}$, then indicates in which half of that interval $\widehat{x}_i$ lies. Thus, if $b_1 = +1$ and $b_2 = -1$, then $\widehat{x}_i \in [0, 1/2]$. Note that some of the bits are not well-defined if $\widehat{x}_i \in B$ where $B = \{-3/4, -1/2, -1/4, 0, 1/4, 1/2, 3/4\}$. Thus, we cannot expect the bit extraction to succeed in this case. In fact, the bit extraction will fail if $\widehat{x}_i$ is sufficiently close to any point in B.

The bit extraction for $\widehat{x}_i$ is performed as follows.

(1) $b_1 \approx T[\widehat{x}_i \times \lceil 1/g(\varepsilon) \rceil]$ (use $G_{\times \lceil 1/g(\varepsilon) \rceil}$-gate)
(2) $\widehat{x}'_i \approx T[\widehat{x}_i - b_1/2]$ (use $G_{-1/2}$-gate and G_+-gate)
(3) $b_2 \approx T[\widehat{x}'_i \times \lceil 1/g(\varepsilon) \rceil]$
(4) $\widehat{x}''_i \approx T[\widehat{x}'_i - b_2/4]$
(5) $b_3 \approx T[\widehat{x}''_i \times \lceil 1/g(\varepsilon) \rceil]$

Note that to compute $-b_1/2$ and $-b_2/4$ we just use the corresponding constant gate, namely $G_{-1/2}$ and $G_{-1/4}$ (with input b_1 or b_2 respectively, instead of const_j). The computation may be incorrect if b_1 or b_2 are not in $\{-1, 1\}$, but in that case the bit-extraction has already failed anyway.

We can show that the bit-extraction succeeds if $\widehat{x}_i$ is sufficiently far away from any point in B. Letting $\text{dist}(t, B) = \min_{p \in B} |t - p|$, we obtain:

Claim 2. *Assume that the circuit-simulator C_j is not corrupted and* $\text{const}_j = +1$. *If* $\text{dist}(T[x_i + j\alpha], B) \geq 8g(\varepsilon)$, *then the bit-extraction phase for $\widehat{x}_i$ outputs the correct bits for $T[x_i + j\alpha]$.*

Phase 3: simulation of λ. Recall that $\lambda : [8]^N \to \{\pm 1, \ldots, \pm N\}$ is the Boolean circuit computing the high-D-Tucker labeling. We interpret $[8]$ as a subdivision of $[-1, 1]$ into standard intervals of length $1/4$. Namely, 1 corresponds to $[-1, -3/4]$, 2 to $[-3/4, -1/2]$, etc. Thus, $[8]^N$ can be interpreted as a subdivision of $[-1, 1]^N$ into hypercubes of side-length $1/4$. With this in mind, we define $\overline{\lambda} : ([-1, 1] \setminus B)^N \to \{\pm 1, \ldots, \pm N\}$, so that for any $x \in ([-1, 1] \setminus B)^N$, $\overline{\lambda}(x)$ is the label that λ assigns to the hypercube containing x.

We can assume that the inputs of λ consist of three bits each, such that the number represented by these three bits yields an element in $[8]$ (by using $[8] \equiv \{0, 1, \ldots, 7\}$). The three bits $b_1, b_2, b_3 \in \{-1, +1\}$ extracted from $T[x_i + j\alpha]$ tell us exactly in which interval $T[x_i + j\alpha]$ lies. Note that if we were to map those bits to $\{0, 1\}$ (where $-1 \mapsto 0$ and $+1 \mapsto 1$), then the bit-string $b_1 b_2 b_3$ would correspond to the number associated with the interval and would thus be the correct corresponding input to the circuit.

We re-interpret the circuit λ as working on bits $\{-1, +1\}$, where -1 corresponds to 0. Clearly, we can implement this circuit in C_j with our Boolean gates. As long as the inputs to every gate are perfect bits (i.e., in $\{-1, +1\}$), the output of the gate will also be a perfect bit, and will correspond to the result of the operation computed by the gate. The inputs to the circuit will be exactly the bits obtained in the bit extraction phase for each $\widehat{x}_i$. Thus, it follows that if the bit extraction phase succeeds, then the simulation of λ will always have a correct output. In other words, using Claim 2, we obtain:

Claim 3. *Assume that the circuit-simulator C_j is not corrupted and* $\text{const}_j = +1$. *If* $\text{dist}(T[x_i + j\alpha], B) \geq 8g(\varepsilon)$ *for all $i \in [N]$, then the simulation phase outputs $\overline{\lambda}(T[x + j\alpha])$.*

Phase 4: output into the Feedback region. For convenience, we will assume that the output of the Boolean circuit λ is encoded in a particular way. It is easy to see that this is without loss of generality, since we can always modify λ so that it follows this encoding. The output of λ is an element in $\{\pm 1, \ldots, \pm N\}$. The encoding we choose uses $2N$ bits $y_1^a, y_1^b, y_2^a, y_2^b, \ldots, y_N^a, y_N^b$ to encode such an element. The element $+i$ is represented by $y_i^a = y_i^b = 1$ and $y_\ell^a = 1$, $y_\ell^b = 0$ for all $\ell \neq i$. The element $-i$ is represented by $y_i^a = y_i^b = 0$ and $y_\ell^a = 0$, $y_\ell^b = 1$ for all $\ell \neq i$.

Recall that in the simulation of λ inside C_j, we actually use the bits $\{-1, +1\}$ instead of $\{0, 1\}$. Thus, output $+i$ is represented by $y_i^a = y_i^b = +1$ and $y_\ell^a = +1$, $y_\ell^b = -1$ for all $\ell \neq i$. Whereas the element $-i$ is represented by $y_i^a = y_i^b = -1$ and $y_\ell^a = -1$, $y_\ell^b = +1$ for all $\ell \neq i$. For any $i \in [N]$ and any $z \in ([-1, 1] \setminus B)^N$ define $\overline{\lambda}_i(z)$ to be:

- $\overline{\lambda}_i(z) = +1$ if $\overline{\lambda}(z) = +i$
- $\overline{\lambda}_i(z) = -1$ if $\overline{\lambda}(z) = -i$
- $\overline{\lambda}_i(z) = 0$ otherwise.

Then, by Claim 3 we obtain that $T[y_i^a + y_i^b] = \overline{\lambda}_i(T[x + j\alpha])$.

In this last phase, for each $i \in [N]$ we compute $T[y_i^a + y_i^b]$ and copy this value into the Feedback region, namely into interval $F_i(j)$ (recall that C_j is the current circuit-simulator). For this we first use a G_+-gate and then a G_{copy}-gate. Thus, we immediately obtain:

Claim 4. *Assume that the circuit-simulator C_j is not corrupted and* $\text{const}_j = +1$. *If* $\text{dist}(T[x_i + j\alpha], B) \geq 8g(\varepsilon)$ *for all* $i \in [N]$, *then* $[C_j(x, \text{const}_j)]_i := v(F_i(j)) = T[\overline{\lambda}_i(T[x + j\alpha]) \pm 2g(\varepsilon)]$ *for all* $i \in [N]$.

3.5 Proof of Correctness

In this section we prove that the reduction works, i.e., from any solution to the Consensus-Halving instance, we can obtain a solution to the original high-D-Tucker instance in polynomial time. In order to do this, we consider two cases and show that we can retrieve a solution in both cases. The first case corresponds to a "well-behaved" solution where there are no stray cuts. The second case corresponds to a solution with stray cuts.

We set $p(n) = 4n^2$ and pick ε such that $16g(\varepsilon) \leq \frac{1}{16p(n)}$, i.e., $\varepsilon \leq \frac{1}{2^{14}n^2}$.

Lemma 3.3. *Let S be any ε-approximate solution for the Consensus-Halving instance. If S does not have any stray cuts, then it yields a solution to the high-D-Tucker instance in polynomial time.*

Lemma 3.4. *Let S be any ε-approximate solution for the Consensus-Halving instance. If S has at least one stray cut, then it yields a solution to the high-D-Tucker instance in polynomial time.*

4 ALGORITHMS FOR SINGLE-BLOCK VALUATIONS

In the previous section, we proved that even when we have 2-block Uniform valuations, the problem remains PPA-hard (even when we are allowed to use $n + n^{1-\delta}$ cuts, for some constant $\delta > 0$). In this section, we will consider the natural case that is not covered by our hardness, that of single-block valuations. Our main results of the section are (a) an algorithm for solving the problem to any precision ε (where ε appears polynomially in the running time), which is parameterized by the maximum number of intersection between the blocks of different agents and (b) a polynomial-time algorithm for 1/2-Consensus-Halving. The latter algorithm generalizes to the case of d-block uniform valuations (in fact, even to the case of piecewise constant valuations with d blocks) if one is allowed to use $d \cdot n$ cuts instead of n. Towards the end of the section, we also present a simple

idea based on linear programming, which allows us to solve the Consensus-Halving problem in polynomial time, when we are allowed to use $2n - \ell$ cuts, for any ℓ which is constant.

4.1 A Parameterized Algorithm for ε-Consensus-Halving

We start with the algorithm for solving ε-Consensus-Halving that is parameterized by the *maximum intersection* between the probability measures. In particular, if d is the maximum number of measures with positive density at any point $x \in [0,1]$ and the maximum value of the densities is at most M then we provide a dynamic programming algorithm that computes an ε-approximate solution in time $O\left(\left(\frac{M}{\varepsilon}\right)^d \cdot \text{poly}(M/\varepsilon, n, d)\right)$. We start with the formal definition of the maximum intersection quantity. In this section we denote by f_i the probability density function of the probability measure μ_i.

Definition 4.1 (Maximum Intersection & Maximum Value). We say that a single-block instance of ε-Consensus-Halving has *maximum intersection* d if for every $x \in [0,1]$ it holds that the set $\mathcal{R}(x) = \{i \in [n] \mid f_i(x) > 0\}$, has cardinality $|\mathcal{R}(x)| \leq d$. We also say that the instance has *maximum value* M if for every $i \in [n]$ and every $x \in [0,1]$ it holds that $f_i(x) \leq M$. This is equivalent to $b_i - a_i \geq 1/M$.

For the rest of the section, we often refer to the following quantity.

Definition 4.2 (Value of Balance). For any ε-Consensus-Halving instance $\mathcal{C} = \{\mu_1, \ldots, \mu_n\}$, any vector of cuts $\vec{s}$ and any $z \in [0,1]$ let $b_i(\vec{s}; z)$ be the mass with respect to μ_i of the part of the interval $[z,1]$ labeled "+" minus the part of the interval labeled "−", when split with the cuts $\vec{s}$. Formally, $b_i(\vec{s}; z) = \mu_i([z,1]^+) - \mu_i([z,1]^-)$, given the set of cuts $\vec{s}$.

The first step of the algorithm is to discretize the interval $[0,1]$ into to intervals of length $\varepsilon/(2M)$. Hence, we split the interval $[0,1]$ in $m = 2M/\varepsilon$ equal subintervals of the form $p_\ell = [(\ell-1)/m, \ell/m]$, where $\ell \in [m]$. The following claim shows the sufficiency of our discretization and follows very easily from the above definitions.

Claim 5. *Let $Q_m = \left\{\frac{\ell-1}{m} \mid \ell \in [m]\right\}$, where $m = M/\varepsilon'$ and let $\mathcal{C} = \{\mu_1, \ldots, \mu_n\}$ be a single-block ε-Consensus-Halving instance with maximum value M. Then we define the* rounded *single-block instance $\mathcal{C}' = \{\mu'_1, \ldots, \mu'_n\}$ where a'_i and b'_i are equal to the number of Q_m that is closer to a_i and b_i respectively. Then every ε-Consensus-Halving solution of $\mathcal{C}'$ is an $(\varepsilon + \varepsilon')$-Consensus-Halving solution of $\mathcal{C}$. Additionally, there exists a solution to the ε'-Consensus-Halving problem where the positions of the cuts lie in the set Q_m.*

Because of Claim 5 we will assume for the rest of this section that we are working with $\mathcal{C}'$ and we will focus on finding $(\varepsilon' = \varepsilon/2)$-Consensus-Halving solution with cuts in Q_m, where $m = 2M/\varepsilon$. This will give us an ε-approximate solution for the single-block instance $\mathcal{C}$. We also need the following definitions:

- for any $z \in [0,1]$ and any instance $\mathcal{C} = \{\mu_1, \ldots, \mu_n\}$ we define the set of measures that have positive mass in $[z,1]$ as follows: $\mathcal{U}(z; \mathcal{C}) = \{i \in [n] \mid \mu_i \in \mathcal{C} \wedge \mu_i([z,1]) > 0\}$ where we might drop $\mathcal{C}$ when it is clear from the context, see also Figure 2,
- for any $z \in [0,1]$ and any instance $\mathcal{C} = \{\mu_1, \ldots, \mu_n\}$ we define the set of measures that have positive density at z as follows: $\mathcal{R}(z; \mathcal{C}) = \{i \in [n] \mid \mu_i \in \mathcal{C} \wedge f_i(z) > 0\}$ where we might drop $\mathcal{C}$ when it is clear from the context, see also Figure 2.

We also define $\bar{Q}_m = \{z \mid z \in Q_m \vee -z \in Q_m\}$. Now we are ready to define our main recursive relation for solving the instance $\mathcal{C}'$. For this we define the function $\alpha(q_1, \ldots, q_d, z, t)$ where $q_j \in \bar{Q}_m$, $z \in Q_m$, and $t \in [n]$ and $\alpha : \bar{Q}_m^d \times Q_m \times [n] \to \{0,1\}$. The intuitive explanation of the value of α is the following:

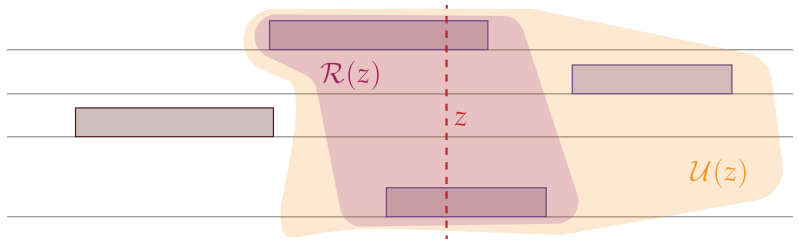

Fig. 2. The definition of the sets $\mathcal{U}(z)$ and $\mathcal{R}(z)$.

$\alpha(q_1, \ldots, q_d, z, t)$ denotes whether it is possible to find t cuts $\vec{s}$ to split the interval $[z, 1]$ such that: (1) if i is the jth element of $\mathcal{R}(z)$ it holds that $|b_i(\vec{s}; z) - q_j| \leq \varepsilon$, (2) for every $i \in \mathcal{U}(z) \setminus \mathcal{R}(z)$ it holds that $|b_i(\vec{s}; 0)| = |b_i(\vec{s}; z)| \leq \varepsilon$.

The value of $\alpha(q_1, \ldots, q_d, z, t)$ can be recursively computed via the following procedure.

- for every $r \in Q_m$, with $r > z$ do
  - if there exists $i \in \mathcal{R}(z)$ and i is the jth element of $\mathcal{R}(z)$ but $i \notin \mathcal{R}(r)$ and by adding the cut r to the set of cuts then the condition $|b_i(r; z) - q_j| > \varepsilon$ holds then **continue**,
  - if there exists $i \in \mathcal{U}(z) \setminus \mathcal{R}(z)$ but $i \notin \mathcal{U}(r) \setminus \mathcal{R}(r)$ then **continue**
  - else for every $i \in \mathcal{R}(r)$, where i is the jth element of $\mathcal{R}(r)$
    - if $i \notin \mathcal{R}(z)$ then we set $q'_j = (-1)^t \mu_i([z, r])$
    - if i is the ℓth element of $\mathcal{R}(z)$ then we set $q'_j = (-1)^t \mu_i([z, r]) + q_\ell$
    - call the function $\alpha(q'_1, \ldots, q'_d, r, t - 1)$
- return **true** if at least one of the recursive calls is successful, and **false** otherwise.

This procedure evaluates the binary function α, but our goal is to solve the search problem that finds a set of cuts that form a solution to ε'-Consensus-Halving. This can be easily done by storing one possible solution whenever $\alpha =$ **true**. We call this possible solution $\beta(q_1, \ldots, q_d, z, t)$. A detailed description of the algorithm is included in the full version.

From the above recursive algorithm we see that the evaluation order for the dynamic programming algorithm that computes $\alpha(q_1, \ldots, q_d, z, t)$ starts from $t = 0$ to $t = n$, $z = 1$ to $z = 0$ and $|q_j| = 1$ to $|q_j| = 0$.

Theorem 4.3. *There exists a dynamic programming algorithm that for any single-block instance C of ε-Consensus-Halving with maximum value M and maximum intersection d, computes a set of cuts that define a solution. The running time of the algorithm is $O\left(\left(\frac{2M}{\varepsilon}\right)^{d+2} n(n+d)\right)$.*

4.2 A polynomial-time algorithm for 1/2-Consensus-Halving

In this subsection, we present a polynomial-time approximation algorithm for 1/2-Consensus-Halving, for the case of single-block valuations. Due to lack of space we present only the high-level description of the algorithm here, and the main theorem. The details are included in the full version.

Theorem 4.4. *There is a polynomial-time algorithm for 1/2-Consensus-Halving, when agents have single-block valuations.*

High-level description: The high-level idea of the algorithm is the following greedy strategy. We will consider the agents in order of non-decreasing height of their valuation blocks. Since each agent has a single block of value, this means that for two agents i and j with $i < j$ that have their value block in intervals I_i and I_j, agent i will be considered before agent j. For each agent, we will attempt to "reserve" a large enough sub-interval of her value block (of total value at least 1/2 for the

agent) and split it in half (to two parts of equal value at least $1/4$ to the agent) using a cut, assigning each half to + and − respectively. At that point, this split will ensure that $|\mu_i(I_i^+) - \mu_i(I_i^-)| \leq 1/2$, regardless of the labeling of the remaining part of the agent's value block I_i. A reserved sub-interval, or *reserved region* (RR) will never be intersected by any subsequent cut. This ensures that the guarantee $|\mu_i(I_i^+) - \mu_i(I_i^-)| \leq 1/2$ for every agent i previously considered will continue to hold.

Reserving a large enough region for the first considered agent is straightforward. For any subsequent agent i, part of her value block interval I_i might already be "covered" by regions that were reserved in previous steps. Before inserting any new cut, we will *expand* some of the RRs which are contained in I_i (those that exhibit a different label on each of their endpoints), until we either ensure that the agent is approximately satisfied with the imbalance of labels that she sees, or these RRs cannot be expanded any longer. In the latter case, we will place the cut corresponding to agent i in I_i, on the midpoint of the "virtual" interval $U_i \subseteq I_i$, consisting of all the intervals of I_i not covered by RRs, "glued" together. Then, we will create a new RR, which will potentially also contain some of the RRs already present in I_i, which will be such that (a) the total value covered by RRs for agent i is $1/2$ and (b) the agent is approximately satisfied (up to a $1/2$) with the value she has for RRs in I_i.

Before we conclude the subsection, we mention that the algorithm can actually be applied to instances with piecewise constant valuations with d blocks, as long as we have $d \cdot n$ cuts at our disposal. The idea is quite simple: Any agent that has (at most) d value blocks will be replaced by (at most) d agents, with single-block valuations. This requires the appropriate scaling of the heights of the blocks, to make sure that the functions are still probability measures. Then, we will run the algorithm, now on $n' \leq d \cdot n$ agents, and we will obtain a solution using n' cuts. This set of cuts will also be a solution to the original instance, since the parameter ε is constant $(1/2)$ in both cases. We obtain the following corollary.

Corollary 4.5. *There exists a polynomial-time algorithm for* $1/2$*-Consensus-Halving, when the agents have* piecewise constant *valuations with* d blocks *and we are allowed to use* $d \cdot n$ *cuts.*

4.3 A Polynomial time Algorithm using $2n - \ell$ cuts

We conclude the section with the following theorem, stating that there is a polynomial-time algorithm for the ε-Consensus-Halving problem with Single-block valuations, if we are allowed to use $2n - \ell$ cuts, for any constant ℓ.

Theorem 4.6. *Let* ℓ *be an integer constant. There exists a polynomial time algorithm that for any single block instance* C *of* ε*-Consensus-Halving computes a set of* $2n - \ell$ *cuts that define a solution. The running time of the algorithm is* $O((2n)^\ell \text{poly}(n, \log(1/\varepsilon)))$.

5 CONSENSUS-1/3-DIVISION IS PPAD-HARD

As we mentioned in the Introduction, our newly developed tools that allowed us to obtain a strengthening of the PPA-completeness result for Consensus-Halving, turn out to be very useful for proving a hardness result for a more general version of the problem, the Consensus-$1/k$-Division problem, for $k = 3$. We provide the definition below.

Definition 5.1 (ε*-Consensus-*$1/k$*-Division).* Let $k \geq 2$. We are given $\varepsilon > 0$ and continuous probability measures $\mu_1, \ldots, \mu_n$ on $[0, 1]$. The probability measures are given by their density functions on $[0, 1]$. The goal is to partition the unit interval into k (not necessarily connected) pieces $A_1, \ldots, A_k$ using at most $(k-1)n$ cuts, such that $|\mu_j(A_i) - \mu_j(A_\ell)| \leq \varepsilon$ for all i, j, ℓ.

For ε-Consensus-1/3-Division, for ease of notation, we will use the labels A/B/C instead of 1/2/3. We state the main theorem of the section.

Theorem 5.2. *ε-Consensus-1/3-Division is PPAD-hard, for inverse-exponential ε.*

As we discussed in the Introduction, the problem for $k \geq 3$ is not necessarily harder than the case of $k = 2$, because we have more cuts at our disposal. Before we present the proof, we highlight some fundamental challenges that arise when one moves from $k = 2$ to larger k.

First, when moving to $k \geq 3$, we move from a simple $+/-$ parity to a more general setting with at least 3 labels. This is already severely problematic when it comes to the reduction of [19], which is highly dependent on the solution having two labels. Indeed, the interpretation of the inputs in [19] and the corresponding design of the gates needs to know which label will appear on the left side of the cut, and special "parity-flip" gadgets are used throughout the reduction to ensure this. On the contrary, with our new value interpretation, we design gadgets which take the label sequence "internally" into account, by adjusting the position of the cut accordingly.

The sequence of the labels however gives rise to a second and more challenging issue: While in the case of $k = 2$ we can assume without loss of generality that the labels between cuts alternate between "+" and "−", we cannot make any such assumptions even when $k = 3$. In fact, it is known that if one restricts the solution to exhibit a cyclic sequence of labels $A/B/C$, then the problem is no longer a total search problem [31]. This seems to be a fundamental obstacle to the design of gates for the case of $k \geq 3$. For $k = 3$, we manage to side-step this obstacle by using a clever "trick": we make sure that the intervals of the Consensus-1/3-Division instance where we read the two inputs (of the 2-dimensional PPAD-complete problem that we reduce from, see Definition 5.6) are placed next to each other, therefore fixing the position of one of the three labels. We prove PPAD-hardness for the *exact* version of the problem (in which we are looking for a perfect balance of the labels, with no allowable discrepancy ε), which will guarantee that in a solution, the value of this label will be fixed to 1/3 throughout the instance. Since our instance is constructed to have piecewise constant valuation functions, the result can be extended to the case of inverse-exponential ε using the following lemma, which is based on an argument of Etessami and Yannakakis [16].

Lemma 5.3. *Let $k \geq 2$. For piece-wise constant valuation functions, exact Consensus-1/k-Division reduces in polynomial time to ε-Consensus-1/k-Division with inverse-exponential ε.*

5.1 A Problem to Reduce From: 2D-Truncated-Linear-FIXP

We start with the following problem, proven to be PPAD-complete by Mehta [29].

Definition 5.4 (2D-Linear-FIXP [29]). The problem 2D-Linear-FIXP is defined as follows. We are given a circuit C using gates $\{+, \times\zeta, \max\}$ and rational constants, that computes a function $F_C : [0,1]^2 \to [0,1]^2$. The goal is to find $x \in [0,1]^2$ such that $F_C(x) = x$.

Theorem 5.5 ([29]). *2D-Linear-FIXP is* PPAD*-complete.*

In order to prove PPAD-hardness of Consensus-1/3-Division, we will use a slightly modified version of 2D-Linear-FIXP, that we call 2D-Truncated-Linear-FIXP. The first difference is that the domain is $[-1,1]^2$ instead of $[0,1]^2$. Furthermore, instead of the gates $\{+, \times\zeta, \max\}$ and rational constants in $\mathbb{Q}$, the circuit will only be allowed to use the gates $\{+_T, \times_T\zeta\}$ and rational constants in $[-1,1] \cap \mathbb{Q}$. The gate $+_T$ corresponds to *truncated addition* and the gate $\times_T\zeta$ corresponds to *truncated multiplication by ζ*. For any $x, y \in [-1,1]$ and any $\zeta \in \mathbb{Q}$, we have $x +_T y = T[x + y]$ and $x \times_T \zeta = T[x \times \zeta]$, where T is the truncation operator in $[-1,1]$ as defined in Section 3.

Definition 5.6 (2D-Truncated-Linear-FIXP). The problem 2D-Truncated-Linear-FIXP is defined as follows. We are given a circuit C using gates $\{+_T, \times_T\zeta\}$ and rational constants in $[-1,1]$,

that computes a function $F_C : [-1,1]^2 \to [-1,1]^2$. The goal is to find $x \in [-1,1]^2$ such that $F_C(x) = x$.

By applying some simple modifications to any 2D-Linear-FIXP circuit, we are able to show the following theorem.

Theorem 5.7. *2D-Truncated-Linear-FIXP is* PPAD*-complete.*

5.2 Description of the Reduction

In order to show that Consensus-1/3-Division is PPAD-hard, we reduce from 2D-Truncated-Linear-FIXP. Namely, given a 2D-Truncated-Linear-FIXP circuit C, we will construct an instance I_C of Consensus-1/3-Division, such that any solution of I_C yields a solution to C (i.e., a fixed-point of $F_C : [-1,1]^2 \to [-1,1]^2$). As before, we will construct a Consensus-1/3-Division instance on some domain $[0, M]$ for some polynomial M, which is easy to transform into an equivalent instance on domain $[0, 1]$.

First, let us give a high-level description of the *ideal* reduction that we would like to construct. First, we would show how any interval of the Consensus-1/3-Division domain encodes a value in $[-1, 1]$. Namely, in any solution S to instance I_C, for every interval I, $v_S(I) \in [-1, 1]$ would be the value encoded by interval I. Then, we would construct agents that implement the arithmetic gates needed by 2D-Truncated-Linear-FIXP, namely $\{+_T, \times_T \zeta\}$ and rational constants in $[-1, 1]$. These agents read some value(s) in $[-1, 1]$ from one or two intervals and output the result of the gate-operation into some other interval of the domain.

With these gates we could implement the circuit C inside our Consensus-1/3-Division instance. In particular, we would have two intervals In_1 and In_2 each representing the two inputs, and two intervals Out_1 and Out_2 each representing the two outputs. These intervals are pairwise disjoint. In the final step we would then "connect" the outputs to the inputs. Namely, we would introduce an agent implementing a $\times_T 1$-gate with input Out_1 and output In_1, and a second agent implementing a $\times_T 1$-gate with input Out_2 and output In_2. This ensures that from any solution of the Consensus-1/3-Division instance we can extract a fixed-point of F_C.

If we could do all this, then this reduction would be very similar to the reduction of Filos-Ratsikas et al. [17] showing that ε-Consensus-Halving is PPAD-hard for constant ε. Unfortunately, there is a significant obstacle. Namely, we don't know how to find an encoding of values in intervals such that we can implement arithmetic gates that always work. Because we don't know in what order the labels A, B and C will appear in any given interval, implementing arithmetic gates is actually much harder than in the case of Consensus-Halving. Thus, the gates we are able to implement only work if the input interval encodes a value in a very specific way. In this case, we say that the interval is a *valid encoding* of a value. Not all intervals will be a valid encoding of a value. In general, it is very hard to enforce valid encodings. This is the reason why our reduction does not seem to generalize to yield hardness for inversely polynomial ε, or to Consensus-$1/k$-Division with $k > 3$.

Nevertheless, for exact Consensus-1/3-Division we are able to find a work-around to force all intervals to be valid encodings of a value. If an interval is a valid encoding of a value (in solution S), then let $v_S(I) \in [-1, 1]$ denote the value encoded by I. We will drop the subscript S in the remainder of the exposition. The following two lemmas are crucial. They are proved in **??**, where the proof of Theorem 5.2 is detailed.

Lemma 5.8. *In the instance I_C we construct, it holds that:*

- *the two intervals In_1 and In_2 are valid encodings, and*
- *if Out_1 and Out_2 are valid encodings, then $v(Out_1) = v(In_1)$ and $v(Out_2) = v(In_2)$.*

Lemma 5.9. *In instance I_C, we can implement arithmetic gates $\{G_{+_T}, G_{\times_T\zeta}, G_\zeta\}$ for operations $\{+_T, \times_T\zeta\}$ and constant-gate respectively, such that:*

- *G_ζ outputs a valid encoding of $\zeta \in [-1,1] \cap \mathbb{Q}$*
- *if the input to $G_{\times_T\zeta}$ is a valid encoding of value $x \in [-1,1]$, then the gate outputs a valid encoding of $x \times_T \zeta$*
- *if the two inputs to G_{+_T} are valid encodings of $x, y \in [-1,1]$, then the gate outputs a valid encoding of $x +_T y$.*

If these two lemmas indeed hold, then the reduction is correct. First of all, by Lemma 5.8, the two inputs to the circuit are valid encodings. Thus, using Lemma 5.9, it follows by induction that all the gates in the circuit perform their operation correctly and output a valid encoding. In particular, the two outputs of the circuit are valid encodings. Thus, by Lemma 5.8, we get that each of the two outputs is equal to the corresponding input. As a result, we have identified a fixed-point of $F_C : [-1,1]^2 \to [-1,1]^2$. The detailed construction is included in the full version.

6 FUTURE DIRECTIONS

The main technical question is whether ε-Consensus-Halving for single-block valuations is PPA-hard or polynomially solvable, or perhaps even complete for some other class. Another interesting direction is to extend the PPA-hardness result of Theorem 3.1 (or even for a larger number of blocks) to constant ε; such a result however would seemingly require some radically new ideas, namely an averaging argument over a constant set of outputs that is robust to stray cuts.

Finally, it would be interesting to study the complexity of the Consensus-$1/k$-Division problem when $k \geq 3$ and possibly strengthen or extend our hardness result to other values of k. To this end, we have recently shown [20] that the problem is in PPA-k, for any k which is a prime power. This begs the question whether Consensus-$1/k$-Division (and consecutively Necklace Splitting with k thieves [18]) is actually complete for PPA-k.

Acknowledgments. Alexandros Hollender is supported by an EPSRC doctoral studentship (Reference 1892947). Katerina Sotiraki is supported in part by NSF/BSF grant #1350619, an MIT-IBM grant, and a DARPA Young Faculty Award, MIT Lincoln Laboratories and Analog Devices. Part of this work was done while the author was visiting the Simons Institute for the Theory of Computing. Manolis Zampetakis is supported by a Google Ph.D. Fellowship and NSF Award #1617730. We would like to thank Paul Goldberg for helpful discussions and comments.

REFERENCES

[1] James Aisenberg, Maria Luisa Bonet, and Sam Buss. 2020. 2-D Tucker is PPA complete. *J. Comput. System Sci.* 108 (2020), 92–103.

[2] Noga Alon. 1987. Splitting necklaces. *Advances in Mathematics* 63, 3 (1987), 247–253.

[3] Noga Alon and Douglas B. West. 1986. The Borsuk-Ulam Theorem and Bisection of Necklaces. *Proc. Amer. Math. Soc.* (1986). https://doi.org/10.2307/2045739

[4] Eshwar Ram Arunachaleswaran, Siddharth Barman, Rachitesh Kumar, and Nidhi Rathi. 2019. Fair and Efficient Cake Division with Connected Pieces. In *Proceedings of the 15th Conference on Web and Internet Economics (WINE)*. Springer, 57–70.

[5] Haris Aziz and Simon Mackenzie. 2016. A discrete and bounded envy-free cake cutting protocol for any number of agents. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, 416–427.

[6] Haris Aziz and Simon Mackenzie. 2016. A discrete and bounded envy-free cake cutting protocol for four agents. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*. 454–464.

[7] Imre Bárány, Senya B. Shlosman, and András Szücs. 1981. On a topological generalization of a theorem of Tverberg. *Journal of the London Mathematical Society* 2, 1 (1981), 158–164.

[8] Karol Borsuk. 1933. Drei Sätze über die n-dimensionale euklidische Sphäre. *Fundamenta Mathematicae* (1933). https://doi.org/10.4064/fm-20-1-177-190

[9] Steven J. Brams and Alan D. Taylor. 1996. *Fair Division: From cake-cutting to dispute resolution.* Cambridge University Press.

[10] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. 2009. Settling the complexity of computing two-player Nash equilibria. *J. ACM* 56, 3 (2009), 14.

[11] Xi Chen, Dimitris Paparas, and Mihalis Yannakakis. 2013. The complexity of non-monotone markets. In *Proceedings of the 45th Annual ACM Symposium on the Theory of Computing (STOC)*. 181–190.

[12] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. 2009. The complexity of computing a Nash equilibrium. In *SIAM Journal on Computing*, Vol. 39. 195–259. https://doi.org/10.1137/070699652

[13] Argyrios Deligkas, John Fearnley, Themistoklis Melissourgos, and Paul G. Spirakis. 2019. Computing Exact Solutions of Consensus Halving and the Borsuk-Ulam Theorem. In *Proceedings of the 46th International Colloquium on Automata, Languages and Programming (ICALP)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 138:1–138:14.

[14] Xiaotie Deng, Jack R. Edmonds, Zhe Feng, Zhengyang Liu, Qi Qi, and Zeying Xu. 2016. Understanding PPA-completeness. In *Proceedings of the 31st Conference on Computational Complexity (CCC)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 23:1–23:25.

[15] Xiaotie Deng, Qi Qi, and Amin Saberi. 2012. Algorithmic solutions for envy-free cake cutting. *Operations Research* 60, 6 (2012), 1461–1476.

[16] Kousha Etessami and Mihalis Yannakakis. 2010. On the complexity of Nash equilibria and other fixed points. *SIAM J. Comput.* 39, 6 (2010), 2531–2597.

[17] Aris Filos-Ratsikas, Søren Kristoffer Still Frederiksen, Paul W. Goldberg, and Jie Zhang. 2018. Hardness Results for Consensus-Halving. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 24:1–24:16.

[18] Aris Filos-Ratsikas and Paul W. Goldberg. 2018. Consensus Halving is PPA-complete. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC)*. ACM, 51–64.

[19] Aris Filos-Ratsikas and Paul W. Goldberg. 2019. The Complexity of Splitting Necklaces and Bisecting Ham Sandwiches. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*. ACM, 638–649.

[20] Aris Filos-Ratsikas, Alexandros Hollender, Katerina Sotiraki, and Manolis Zampetakis. 2020. A Topological Characterization of Modulo-p Arguments and Implications for Necklace Splitting. *arXiv preprint arXiv:2003.11974* (2020).

[21] Jugal Garg, Ruta Mehta, and Vijay V. Vazirani. 2018. Substitution with Satiation: A New Class of Utility Functions and a Complementary Pivot Algorithm. *Mathematics of Operations Research* 43, 3 (2018), 996–1024.

[22] Charles H. Goldberg and Douglas B. West. 1985. Bisection of Circle Colorings. *SIAM Journal on Algebraic Discrete Methods* (1985). https://doi.org/10.1137/0606010

[23] Paul W. Goldberg and Alexandros Hollender. 2019. The Hairy Ball Problem is PPAD-Complete. In *Proceedings of the 46th International Colloquium on Automata, Languages and Programming (ICALP)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 65:1–65:14.

[24] Mika Göös, Pritish Kamath, Katerina Sotiraki, and Manolis Zampetakis. 2019. On the Complexity of Modulo-q Arguments and the Chevalley-Warning Theorem. In *Proceedings of the 35th Conference on Computational Complexity (CCC) (to appear)*.

[25] Michelangelo Grigni. 2001. A Sperner lemma complete for PPA. *Inform. Process. Lett.* (2001). https://doi.org/10.1016/S0020-0190(00)00152-6

[26] Charles R. Hobby and John R. Rice. 1965. A moment problem in L1 approximation. *Proc. Amer. Math. Soc.* 16, 4 (1965), 665–670.

[27] Alexandros Hollender. 2019. The Classes PPA-k: Existence from Arguments Modulo k. In *Proceedings of the 15th Conference on Web and Internet Economics (WINE)*. Springer, 214–227.

[28] Nimrod Megiddo and Christos H. Papadimitriou. 1991. On total functions, existence theorems and computational complexity. *Theoretical Computer Science* (1991). https://doi.org/10.1016/0304-3975(91)90200-L

[29] Ruta Mehta. 2014. Constant rank bimatrix games are PPAD-hard. In *Proceedings of the 46th Annual ACM Symposium on the Theory of Computing (STOC)*. 545–554.

[30] Jerzy Neyman. 1946. Un theoreme d'existence. *CR Acad. Sci. Paris* 222 (1946), 843–845.

[31] Dömötör Pálvölgyi. 2009. Combinatorial Necklace Splitting. *The Electronic Journal of Combinatorics* 16(1), R79 (2009), 1–8.

[32] Christos H. Papadimitriou. 1994. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. System Sci.* 48, 3 (1994), 498–532.

[33] Forest W. Simmons and Francis Edward Su. 2003. Consensus-halving via theorems of Borsuk-Ulam and Tucker. *Mathematical social sciences* 45, 1 (2003), 15–25.

[34] Hugo Steinhaus. 1948. The Problem of Fair Division. *Econometrica* 16 (1948), 101–104.

[35] Albert W. Tucker. 1945. Some Topological Properties of Disk and Sphere. In *Proceedings of the First Canadian Math. Congress, Montreal*. University of Toronto Press, 286–309.