Constant Inapproximability for PPA

Constant Inapproximability for PPA*

Argyrios Deligkas
Royal Holloway, United Kingdom
argyrios.deligkas@rhul.ac.uk

John Fearnley
University of Liverpool, United Kingdom
john.fearnley@liverpool.ac.uk

Alexandros Hollender
University of Oxford, United Kingdom
alexandros.hollender@cs.ox.ac.uk

Themistoklis Melissourgos
University of Essex, United Kingdom
themistoklis.melissourgos@essex.ac.uk

Abstract

In the ε-Consensus-Halving problem, we are given n probability measures $v_1, \ldots, v_n$ on the interval $R = [0, 1]$, and the goal is to partition R into two parts R^+ and R^- using at most n cuts, so that $|v_i(R^+) - v_i(R^-)| \leq \varepsilon$ for all i. This fundamental fair division problem was the first natural problem shown to be complete for the class PPA, and all subsequent PPA-completeness results for other natural problems have been obtained by reducing from it.

We show that ε-Consensus-Halving is PPA-complete even when the parameter ε is a constant. In fact, we prove that this holds for any constant $\varepsilon < 1/5$. As a result, we obtain constant inapproximability results for all known natural PPA-complete problems, including Necklace-Splitting, the Discrete-Ham-Sandwich problem, two variants of the pizza sharing problem, and for finding fair independent sets in cycles and paths.

*A preliminary version of this paper appeared at STOC '22.

1 Introduction

The consensus halving problem [Simmons and Su, 2003] is a fair division problem defined by n agents, who each have a valuation function over the unit interval $R = [0, 1]$. The goal is to partition R into two sets R^+ and R^- using at most n cuts, such that all agents agree that R^+ and R^- have the same valuation, or in the ε-approximate version, that all agents agree that R^+ and R^- have valuations that differ by at most ε.

The problem is guaranteed to have a solution and this is usually proved by using the Borsuk-Ulam theorem from topology, or its discrete counterpart, Tucker's lemma [Simmons and Su, 2003]. In fact, very similar versions of this existence result have been proved in the past in different contexts [Hobby and Rice, 1965; Alon and West, 1986; Alon, 1987]. Since the problem is guaranteed to have a solution and solutions can be verified efficiently, it lies in the complexity class TFNP: the class of total NP search problems. In particular, this means that the problem cannot be NP-hard, unless NP = co-NP [Megiddo and Papadimitriou, 1991], and instead, one has to use subclasses of TFNP to classify its complexity.

The consensus halving problem has risen to prominence as it has played a crucial role in the development of the complexity class PPA, a subclass of TFNP defined by Papadimitriou [1994]. Indeed, in a breakthrough result, Filos-Ratsikas and Goldberg [2018] proved that the problem is complete for PPA. This was the first "natural" complete problem for the class and it has been pivotal in proving further such completeness results. For example, PPA-completeness has since been shown for other "natural" problems such as the necklace splitting problem and the discrete ham sandwich problem [Filos-Ratsikas and Goldberg, 2019], two types of the pizza-sharing problem [Deligkas et al., 2022a; Schnider, 2022], and finding fair independent sets in cycles and paths [Haviv, 2021]. We refer to these as natural problems[1] since their definition does not involve any kind of circuit, as opposed to "unnatural" problems like Tucker (the problem associated with Tucker's Lemma), which was already known to be PPA-complete [Aisenberg et al., 2020], but whose definition involves a Boolean circuit.

Consensus halving has been used in a fundamental way to show PPA-completeness for natural problems, because it bridges the gap between natural and unnatural PPA-complete problems. Specifically, the PPA-hardness results for consensus halving [Filos-Ratsikas and Goldberg, 2018, 2019] reduce from Tucker, and explicitly remove the Boolean circuit by encoding each gate as a consensus halving agent. To the best of our knowledge, *all* subsequent hardness results for natural problems have reduced from consensus halving.

Hardness of approximation. Prior work has shown that, not only is it PPA-complete to find exact consensus halving solutions for piecewise constant valuation functions, but it is also PPA-complete to find approximate solutions. The initial hardness result of Filos-Ratsikas and Goldberg [2018] showed that ε-Consensus-Halving is PPA-complete for an exponentially small ε (in the size of the input). The same authors later improved this to obtain a PPA-completeness result for ε-Consensus-Halving with ε being polynomially small [Filos-Ratsikas and Goldberg, 2019].

The hardness of approximation for consensus halving has then directly led to hardness of approximation for the other natural PPA-complete problems, because all of the PPA-hardness reductions for natural problems that have been discovered so far preserve

[1] We note that some of these problems also have more general "unnatural" versions where the inputs, e.g., the valuations, are represented by circuits. The aforementioned completeness results apply to the natural versions without circuits.

approximate solutions. So, we have that necklace splitting, discrete ham sandwich, pizza-sharing, and finding fair independent sets in cycles and paths are all PPA-complete to approximate for a polynomially small ε.

In this sense, consensus halving plays a crucial role in the hardness of approximation for natural PPA-complete problems, because any improvement in the hardness result for consensus halving directly leads to an improvement in the hardness results for *all* of the natural problems that are currently known to be PPA-complete.

The key question left open by previous work is whether ε-Consensus-Halving is PPA-hard, and thus PPA-complete, for a constant ε. While there is no such result in prior work, consensus halving is known to be PPAD-hard for a very small constant ε [Filos-Ratsikas et al., 2018]. This result actually predates all of the PPA-hardness results and arises from a direct reduction to ε-Consensus-Halving from the Gcircuit problem, which is known to be PPAD-complete for constant ε [Rubinstein, 2018]. Notably, though, the constant is so small that no prior work has actually given a lower bound on its magnitude.

Furthermore, even ignoring the minuscule ε, this result is somewhat unsatisfying, since it seems unlikely that PPAD-hardness is the correct answer for constant approximation, given that PPAD $\subseteq$ PPA, and PPA appears to capture a strictly larger class of problems. This is doubly so, since finding a polynomially small approximation is known to be PPA-complete, and thus PPA-completeness of finding constant approximations would be the natural, and tight, answer.

Our Contribution. Our main result is as follows.

Theorem 1.1. *ε-Consensus-Halving is PPA-complete for all $\varepsilon < 1/5$.*

Thus, we show hardness for a constant ε, improving upon the prior state-of-the-art result, which showed hardness for a polynomially small ε. A direct consequence of this theorem is that the hardness results for *all* natural problems that are known to be PPA-complete are strengthened as well, and we obtain PPA-hardness for each of the problems for a constant ε.

Our result shows hardness for any $\varepsilon < 1/5$, which is notably large compared to other constant inapproximability results for total search problems. For example, the current state-of-the-art hardness results for PPAD-complete problems do show hardness for a constant ε [Rubinstein, 2018], but as mentioned earlier, that constant is so small that no prior work has given a lower bound on its magnitude.[2] Here we give a constant that is substantial relative to the trivial upper bound of $\varepsilon = 1$. We obtain similarly large constants for each of the other natural problems that are known to be PPA-complete, as shown in the following table.

Problem	PPA-completeness threshold
ε-NecklaceSplitting	1/5
ε-DiscreteHamSandwich	1/5
ε-StraightPizzaSharing	1/5
ε-SquarePizzaSharing	1/5
ε-FairSplitPath	1/20
ε-FairSplitCycle	1/20

[2] In subsequent work, we obtained similar strong explicit inapproximability bounds for PPAD problems [Deligkas et al., 2024].

The full details of these follow-on hardness results can be found below in Section 1.1.

Moreover, our main result continues to hold even if we severely restrict the valuation functions of the agents.

Theorem 1.2. *ε-Consensus-Halving is PPA-complete for all $\varepsilon < 1/5$, even if all agents have 3-block uniform valuations.*

An agent has a 3-block uniform valuation function if the density function of the valuation is non-zero in at most three intervals, and in each such interval it has the same non-zero value.

Finally, by a standard argument [Filos-Ratsikas et al., 2023], it immediately follows that the hardness result holds even if we allow a few more than just n cuts.

Theorem 1.3. *ε-Consensus-Halving is PPA-complete for all $\varepsilon < 1/5$, even if all agents have 3-block uniform valuations, and even if $n + n^{1-\delta}$ cuts are allowed for some constant $\delta \in (0, 1]$, where n is the number of agents.*

For completeness we provide a proof of this result in Appendix A.

1.1 Direct Consequences

Our hardness result for ε-Consensus-Halving directly yields improved hardness results for every natural problem that is currently known to be PPA-complete. In this section we give the details for these improved hardness results.

Necklace Splitting. In NecklaceSplitting, we are given a necklace with beads of n colours, and we want to split the necklace into two (in general, non-contiguous) parts by making at most n cuts, such that both parts contain half of the beads of each colour. It was shown by Goldberg and West [1985] and Alon and West [1986] that the problem always admits a solution, and later Alon [1987] extended this result to the variant where the necklace must be divided into k parts rather than two.

PPA-completeness for the problem was proven by Filos-Ratsikas and Goldberg [2019] via a reduction from ε-Consensus-Halving for an inversely-polynomial ε. In addition, in [Filos-Ratsikas and Goldberg, 2018] it was proven that the approximate version of the problem is PPAD-hard for some small constant ε.

In the approximate version of the problem, denoted as ε-NecklaceSplitting with $\varepsilon \in (0, 1)$, the goal is to cut the necklace into two parts such that, for each colour, the discrepancy between the two parts is bounded by ε. Formally, if there are B_i beads of colour i and B_i^+, B_i^- correspond to the number of beads of colour i in each of the two parts, in an ε-solution it holds that $|B_i^+ - B_i^-| \leq \varepsilon \cdot B_i$. The reduction presented in [Filos-Ratsikas and Goldberg, 2018] increases the error of the ε-Consensus-Halving instance by only a polynomially small amount[3], so by applying our our main result, we obtain the following.

Theorem 1.4. *ε-NecklaceSplitting is PPA-complete for every constant $\varepsilon < 1/5$, even if $n + n^{1-\delta}$ cuts are allowed for some constant $\delta > 0$.*

[3]We note that [Filos-Ratsikas and Goldberg, 2018] have defined ε-NecklaceSplitting with ε denoting the discrepancy between the number of beads in each of the two parts, rather than normalising ε so that it is expressed relative to the total number of beads. However, this appears to have been a mistake, since their proof and result actually use the definition that we give here.

Ham Sandwich. In DiscreteHamSandwich, as defined by Papadimitriou [1994], we are given n sets of points with integer coordinates in d dimensional space, where $d \geq n$. The task is to find a hyperplane that cuts the space into two halfspaces, such that each halfspace contains half of the points of each set. If any points lie on the plane, then we are allowed to place each of them on either side. [Filos-Ratsikas and Goldberg, 2019] proved that the problem is PPA-complete, via a reduction from NecklaceSplitting.

In the approximate version of the problem, denoted ε-DiscreteHamSandwich, we want to find a hyperplane such that, for every set, the discrepancy between the number of points contained in the two halfspaces is bounded by ε. Formally, if there are S_i points for set i and S_i^+, S_i^- correspond to the number of points belonging to the two halfspaces, in an ε-solution we must have $|S_i^+ - S_i^-| \leq \varepsilon \cdot S_i$. The reduction between DiscreteHamSandwich and NecklaceSplitting presented by Filos-Ratsikas and Goldberg [2019] is approximation preserving, so we get the following theorem.

Theorem 1.5. *ε-DiscreteHamSandwich is PPA-complete for every constant $\varepsilon < 1/5$, even if $d = n + n^{1-\delta}$ for some constant $\delta > 0$.*

Pizza Sharing. In pizza sharing problems we are given measurable objects that are embedded in the two-dimensional plane, and we are asked to make a number of cuts in order to divide each mass into two equally sized portions. Two versions of this problem have been studied in the literature.

In the StraightPizzaSharing problem, we are given $2n$ two-dimensional masses in the plane, and we are asked to find ℓ straight lines that create a "checkerboard" that simultaneously bisects all of the masses. In [Barba et al., 2019; Hubard and Karasev, 2020] it was shown that the problem always admits a solution when $\ell \geq n$.

In the SquarePizzaSharing problem, there are n masses in the plane, and the task is to simultaneously bisect all masses via a *square-cut*: a path that is the union of horizontal and vertical line segments. In [Karasev et al., 2016] it was proven that a path with $n-1$ turns can always bisect all n masses.

In the approximate versions of these problems, we are looking for an approximate bisection. Formally, in an ε-approximate solution of these problems, we are looking for a partition of the plane R into two regions R^+ and R^- such that for every measure μ_i it holds that $|\mu_i(R^+) - \mu_i(R^-)| \leq \varepsilon \cdot \mu_i(R)$. For ε-StraightPizzaSharing the partition must be produced by ℓ lines, while for ε-SquarePizzaSharing the partition must be produced by a square-cut path with t turns.

Both problems were proven to be PPA-complete when ε is inversely polynomial and PPAD-hard for a small constant $\varepsilon \in (0, 1)$ via direct reductions from Consensus-Halving [Deligkas et al., 2022a; Schnider, 2022]. Using the reductions from [Deligkas et al., 2022a], which increase the error by at most a polynomially small amount, alongside our main theorem yields the following.

Theorem 1.6. *ε-StraightPizzaSharing is PPA-complete for every constant $\varepsilon < 1/5$, even if $n + n^{1-\delta}$ cuts are allowed for some constant $\delta > 0$.*

Theorem 1.7. *ε-SquarePizzaSharing is PPA-complete for every constant $\varepsilon < 1/5$, even if the square-cut path is allowed to have $n + n^{1-\delta}$ turns for some constant $\delta > 0$.*

Fair Independent Sets. In this setting, we are given a graph G whose vertices are partitioned into n sets $V_1, \ldots, V_n$. The task is to find two independent sets of G such that every V_i is covered in a "fair" manner. In particular, we are interested in the setting

where G is a cycle or a path, since it was proven that such graphs possess fair independent sets [Aharoni et al., 2017; Alishahi and Meunier, 2017; Black et al., 2020].

More specifically, Alishahi and Meunier [2017] proved that if G is a cycle of m vertices and n has the same parity as m, then there exist two disjoint independent sets S_1 and S_2, such that for every $i \in [n]$ it holds that $|V_i \cap (S_1 \cup S_2)| = |V_i| - 1$ and $|S_j \cap V_i| \geq \frac{1}{2}|V_i| - 1$ for all $j \in \{1, 2\}$. For $\varepsilon \in [0, \frac{1}{2}]$, we use ε-FairSplitCycle to denote the problem of finding two such independent sets, where the second condition is relaxed to $|S_j \cap V_i| \geq (\frac{1}{2} - \varepsilon) \cdot |V_i| - 1$.

A similar theorem was shown for paths by Black et al. [2020]: if G is a path and every set V_i contains an odd number of points, then there exist two independent sets S_1 and S_2, covering all but at most n vertices of G such that for every $i \in [n]$ it holds that $|S_1 \cap V_i| \in \left[\frac{1}{2}|V_i| - 1, \frac{1}{2}|V_i|\right]$. For $\varepsilon \in [0, \frac{1}{2}]$, we use ε-FairSplitPath to denote the corresponding computational problem, where the condition is relaxed to $|S_1 \cap V_i| \in \left[(\frac{1}{2} - \varepsilon) \cdot |V_i| - 1, (\frac{1}{2} + \varepsilon) \cdot |V_i|\right]$.

Hardness was shown for both problems by Haviv [2021], who proved that both problems are PPA-complete for a polynomially small ε and that they are PPAD-hard for a small constant ε. The hardness is shown by a reduction from ε-Consensus-Halving to $\frac{\varepsilon}{4}$-FairSplitPath and then a follow-on reduction from ε-FairSplitPath to ε-FairSplitCycle. Combining these reductions with our main theorem yields the following.[4]

Theorem 1.8. *ε-FairSplitPath and ε-FairSplitCycle are PPA-complete for every constant $\varepsilon < 1/20$.*

1.2 Further Related Work

There are various other works that relate to ours, some of which dealt with PPA-hardness and some of which studied the consensus halving problem.

In a recent work by Deligkas et al. [2022c] the main question was "How does the complexity of consensus halving depend on the number of agents?". This paper's main result is a dichotomy between 2 and 3 agents when the valuations are monotone (but possibly non-additive). In particular, for the former case the problem is polynomial time solvable, while for the latter it is PPA-complete. If the monotonicity property is dropped, then both cases become PPA-complete. Furthermore, for the case of a single agent (and even for n agents with identical valuations) the problem is polynomial time solvable.

Alon and Graur [2021] present a set of strong positive results on the ε-NecklaceSplitting problem. They present efficient algorithms for a relaxed version of this problem where more than n cuts are allowed. In particular, for an instance whose beads can take n colours, and can be at most m per colour, they give an offline and an online algorithm that is efficient and deterministic, which provide a solution by making at most $O(n(\log m + O(1)))$ and $O(m^{2/3} \cdot n(\log n)^{1/3})$ cuts, respectively, for $\varepsilon = 0$. For $\varepsilon > 0$, the same algorithms work with the aforementioned number of cuts, by substituting m with $1/\varepsilon$. These algorithms also work for the ε-Consensus-Halving problem when we are allowed to use more than n cuts. Their positive results extend to the generalization of NecklaceSplitting in which, instead of wishing to split each colour's beads into two parts, we split them into $k \geq 2$ parts [Alon, 1987]. For detailed definitions of this and related problems, as well as their related complexity classes, see [Filos-Ratsikas et al., 2021], and [Hollender, 2021].

In [Goldberg et al., 2022] the problem under study deviates slightly from the typical consensus halving problem. There are (divisible) items and they are not presented in a

[4] In a subsequent version of his work, Haviv [2022] improved the parameters of his reduction from Consensus-Halving to FairSplitPath, and thus obtained PPA-hardness results for $\varepsilon < 1/10$.

linear order, but rather unordered, with agents having linear and additively separable utilities over them. In this work the authors provide polynomial time algorithms even for the more general Consensus-Halving problem where we do not split the probability measures in two, but in $k \geq 2$ parts, and show that for a slightly non-linear valuation class the problem becomes PPAD-hard. For the case where the items are in a specific order, they show that the problem is PPA-complete.

There is also a hierarchy of complexity classes of problems that seek exact solutions, whose output involves irrational numbers. A famous such class is FIXP whose typical problem has as input a function from $[0,1]^N$ to itself, and the task is to find an exact Brouwer fixed point of the function. This class was defined by Etessami and Yannakakis [2010] who further showed that the problem of finding an exact Nash equilibrium for $n \geq 3$ agents is complete for the class. In [Deligkas et al., 2021] a related class was defined, namely BU, whose typical problem has as input a function from the L_1 unit N-sphere to $\mathbb{R}^N$, and the task is to find an exact Borsuk-Ulam point, i.e., one that has the same function value as its antipodal. This work showed that exact Consensus-Halving is in BU for piecewise polynomial valuations and FIXP-hard. Both the aforementioned papers showed that when the input function is piecewise linear, then the induced class is identical to PPAD and PPA respectively. Etessami and Yannakakis [2010] also defined the strong approximation version of an exact search problem where a point that is close to an exact solution is sought. By extending this notion to BU, Batziou et al. [2021] showed that the strong approximation version of Consensus-Halving (with valuations represented by algebraic circuits) is complete for the corresponding class BU_a.

2 Preliminaries

A *valuation function*, or simply valuation, of an agent is a probability measure over the interval $R = [0,1]$. The probability measures are given by their density functions. A valuation function is *piecewise constant* if R can be partitioned into a finite set of intervals where the density of the probability is constant over each interval. Thus, a piecewise constant valuation can be explicitly represented as endpoints and heights of value blocks. For any measurable subset S of R, $v_i(S)$ denotes the value of agent i for S; $v_i(S)$ equals the measure of the probability of agent i over S. In particular, $v_i(R) = 1$. For $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$.

Definition 1 (ε-Consensus-Halving)**.** An instance of ε-Consensus-Halving consists of n agents with piecewise constant valuation functions over the interval $R = [0,1]$. A solution is a partition of the interval R into two regions R^+ and R^-, using at most n cuts, where every agent agrees that the value of R^+ is at most ε-away from the value for R^-. Formally, in a solution of ε-Consensus-Halving it holds that $|v_i(R^+) - v_i(R^-)| \leq \varepsilon$ for every $i \in [n]$.

In this paper we will show a hardness result for ε-Consensus-Halving by reducing from the 2D-Tucker problem.

Definition 2 (2D-Tucker)**.** An instance of 2D-Tucker consists of a labelling function $\lambda : [m] \times [m] \to \{\pm 1, \pm 2\}$ such that for $1 \leq i, j \leq m$, $\lambda(i,1) = -\lambda(m - i + 1, m)$ and $\lambda(1,j) = -\lambda(m, m - j + 1)$. A solution to such an instance is a pair of vertices (x_1, y_1), (x_2, y_2) with $|x_1 - x_2| \leq 1$ and $|y_1 - y_2| \leq 1$ such that $\lambda(x_1, y_1) = -\lambda(x_2, y_2)$.

The labelling λ is given as a Boolean circuit. 2D-Tucker is known to be PPA-complete; Papadimitriou [1994] proved membership in PPA and Aisenberg et al. [2020] proved PPA-hardness.

Theorem 2.1 (Aisenberg et al. [2020]; Papadimitriou [1994])**.** 2D-Tucker *is* PPA*-complete.*

Other versions of Tucker's lemma have also been shown to be PPA-complete [Deng et al., 2017; Filos-Ratsikas and Goldberg, 2019].

3 Technical Overview

In this section, we present an overview of the proof of our main result including the new insights that allow us to obtain hardness for a constant ε.

To prove our main result, we reduce 2D-Tucker to ε-Consensus-Halving for all $\varepsilon < 1/5$. Here we give an overview of the reduction, and the key challenges that needed to be overcome in order to obtain a constant ε.

In our description of the main ideas and challenges, we will make reference to the three existing PPA-hardness reductions for consensus halving.[5]

- **Work 1** [Filos-Ratsikas and Goldberg, 2018]: which proves hardness for inverse exponential ε.
- **Work 2** [Filos-Ratsikas and Goldberg, 2019]: which proves hardness for inverse polynomial ε.
- **Work 3** [Filos-Ratsikas et al., 2023]: which provides a significantly simplified proof of hardness for inverse polynomial ε.

All three existing works ultimately reduce from 2D-Tucker, but Works 2 and 3 include a preliminary step, where 2D-Tucker is reduced to its high-dimensional version: ND-Tucker. This seems to be necessary in order to obtain hardness for inverse polynomial ε. Indeed, a similar observation can also be made about analogous results in the study of approximate Nash equilibrium computation [Daskalakis et al., 2009; Chen et al., 2009], where a high-dimensional version of the Brouwer problem is used to achieve hardness for inverse polynomial approximation.

We begin with a very high-level overview of the general structure of the reduction which applies to all three existing works, as well as to ours. Informally, an ND-Tucker instance is defined over an N-dimensional grid $G = [m] \times [m] \times \cdots \times [m]$ with side length m. The instance gives a *labelling function* $\lambda : G \to \{-N, \ldots, -1, 1, \ldots, N\}$, presented as a Boolean circuit, that assigns each point in the grid a label that is either $+i$ or $-i$ for some i in the range $1 \leq i \leq N$. Additionally, the labelling satisfies an *antipodality* condition on the boundary: letting $\overline{x_i} := m - x_i + 1$, it holds that $\lambda(\overline{x}) = -\lambda(x)$ whenever x lies on the boundary of G. The goal is to find two points x and y on the grid, such that x and y are within L_∞ distance 1 of each other, and $\lambda(x) = -\lambda(y)$. Such a pair of points is guaranteed to exist by Tucker's Lemma [Tucker, 1945], and the problem of finding one is PPA-complete even for constant m, as shown in Work 2 by reducing from 2D-Tucker. The problem 2D-Tucker is defined in the same way, except that $N = 2$, and m is required to be exponentially large for the problem to be PPA-complete [Aisenberg et al., 2020].

We are now ready to present the high-level setup used in all three previous works. The specifics of the reductions in Works 1 and 2 are significantly more involved than what is presented here, and so the presentation below should be seen as mostly applying to the

[5]The paper [Filos-Ratsikas and Goldberg, 2023] is a journal version, which combines the proof from Work 2 (which reuses some machinery from Work 1) with some results from Work 1, such as the connection between Consensus-Halving and NecklaceSplitting.

simplified proof of Work 3 (while still representing the underlying core structure hidden behind the reductions in Works 1 and 2). The ε-Consensus-Halving instance CH(λ) is constructed as follows:

- The line $R = [0, 1]$ consists of two intervals I and C. We think of I as the *input region* (also called *coordinate-encoding region* in prior work), and C as the *circuit region* (also called *circuit-encoding region*). In any solution $S = (R^+, R^-)$ to the instance CH(λ), I will be partitioned into two regions $I^+ := I \cap R^+$ and $I^- := I \cap R^-$. The exact way in which I is partitioned encodes a point $z := z(I^+, I^-)$ in some domain. In Work 1, this domain is a locally two-dimensional Möbius strip, while in Work 2, it is a high-dimensional generalization of that. Work 3 significantly simplifies this encoding by letting the domain simply be the N-dimensional unit hypercube. In all three cases, the grid G of the ND-Tucker instance λ is embedded in the domain in question, and so the partition (I^+, I^-) ultimately encodes a point $x := x(I^+, I^-) \in G$.

- To "extract" the point $x \in G$ from (I^+, I^-), a binary decoding step is performed where the continuous information that is encoded in (I^+, I^-) is converted into bit values that represent x_i. Mechanically, this is implemented by introducing a set of agents in CH(λ) who ensure that cuts (between R^+ and R^-) are placed in specified regions of $C \subset [0, 1]$ to encode either a 0 bit or a 1 bit. In Works 1 and 2, this binary decoding is performed "at the source", namely the information read from I is essentially binary, and then further processed by simulating Boolean gates. In Work 3, the information read from I is continuous and then further processed by simulating arithmetic gates. The arithmetic gates are then used to perform the bit decoding.

- As is common in PPAD and PPA reductions, the Boolean decoding step from z to x can fail for certain values, and if the decoding step fails then nonsensical values will be produced. To address this, instead of just decoding z, a large number of points surrounding the encoded point z are decoded, where the samples are chosen to ensure that only a small number of the decoding steps fail. If K samples are taken, then this gives a sequence of points $x^1, x^2, \ldots, x^K$, most of which are valid bit representations of points in G, and a small number of which contain nonsensical values. Importantly, the sampling is performed such that all resulting (correctly) decoded points in G lie within L_∞ distance 1 of each other.

- The next step is to simulate the execution of the Tucker labelling circuit λ on the inputs $x^1, x^2, \ldots, x^K$. For this, K completely independent copies of the circuit λ are simulated, each within its own sub-region $C^1, \ldots, C^K$ of the circuit region C. The region C^k is fed input x^k and so is used to compute $\lambda(x^k)$. Mechanically, each gate in each circuit is simulated by a set of agents who ensure that the output of each gate is encoded by a cut in a specified region of C^k, allowing other agents to read that value to simulate other gates.

- The output of the circuit in C^k is treated as a vector $y^k \in [-1, 1]^N$, so that whenever x^k was correctly decoded and $\lambda(x^k) = +i$ (resp. $-i$), we have $y^k_i = +1$ (resp. -1) and $y^k_\ell = 0$ for all other dimensions $\ell \neq i$. If x^k was not correctly decoded, then y^k can be any vector in $[-1, 1]^N$.

- The last step is to average the outputs of the circuits. Specifically, for each dimension $i \in [N]$, we introduce an agent that computes the average $L(i) = \frac{1}{K} \sum_{k=1}^{K} y^k_i$ and enforces that $L(i)$ be ε-close to zero for all i. With K being chosen to be suitably large, the effect of the incorrectly decoded points becomes negligible, and so it is

only possible for $L(i)$ to be close to zero for all i if, for each dimension i, there are a roughly equal number of points with label $+i$ and label $-i$. Since all of the input points lie within L_∞ distance 1 of each other, this implies that we have a solution to the Tucker instance, namely, we can extract from $x^1, \ldots, x^K$ two points yielding a solution to ND-Tucker. Mechanically, this is implemented by a set of N agents, one for each dimension i, enforcing that $L(i)$ is close to zero for a specific label i.

- The final – and crucial – complication that significantly differentiates these reductions from more standard PPAD-hardness reductions is the presence of *stray cuts*. Indeed, the construction we have described works perfectly assuming that there are N cuts in the input region I. However, nothing forces these N cuts to be made in the input region. If there are less than N cuts in the input region, then we think of the missing cuts as having become *stray cuts* that can interfere with the rest of the construction, in particular the various circuit simulations. Indeed, a stray cut can essentially destroy the output of one of the circuit simulations by occurring in the corresponding region C^k. Fortunately, this is easy to fix by taking enough additional samples and copies of the circuit.

  The more problematic – and conceptually important – interference caused by stray cuts is that a single stray cut can influence any fraction of the circuits (for example, half of them) by "changing their perception of whether a bit is 1 or 0." Intuitively, any solution $S = (R^+, R^-)$ of $\mathrm{CH}(\lambda)$ remains a solution if we flip $+$ and $-$, i.e., if we let $S' = (R^-, R^+)$. This symmetry has the following important consequence: in order to perform a logical operation such as AND, which does not commute with bit-flipping (i.e., $\mathrm{AND}(\neg b_1, \neg b_2) \neq \neg\mathrm{AND}(b_1, b_2)$), the circuit needs to be given access to some ground-truth value. This ground-truth essentially helps the circuit differentiate between bits 1 and 0, and thus allows it to implement logical gates such as AND. If there are no stray cuts, then it is not too hard to ensure that all the copies of the circuit see the same ground-truth. But a single stray cut can change the ground-truth perception of half the circuits. By a careful construction, it is possible to ensure that if circuit C^k's perception of the ground-truth is altered by a stray cut, then it will output $-\lambda(\overline{x^k})$ instead of $\lambda(x^k)$. Furthermore, the construction ensures that if there are less than N cuts in I, and thus at least one stray cut, then all the correctly decoded points amongst $x^1, \ldots, x^K$ lie on the boundary of G. Using the antipodality conditions of λ, it follows that $-\lambda(\overline{x^k}) = \lambda(x^k)$ for all valid points x^k, and thus the difference in ground-truth between the circuit copies does not matter anymore.

  In a certain sense, stray cuts are necessary for any reduction proving PPA-hardness for the problem. For example, if there was some trick to enforce that N cuts lie in I in the reduction above, then the reduction would not have made use of the antipodality condition of λ. This is not possible, since we could then reduce from a circuit λ that has no solution, but $\mathrm{CH}(\lambda)$ always has a solution. More generally, it can be shown that any reduction where each cut has its own disjoint reserved region (where it must lie) can only prove PPAD-hardness at best. Indeed, in that case those instances can be reduced to the problem of finding a Brouwer fixed point.

Our reduction follows this basic template, but requires overcoming various challenges in order to obtain a constant ε.

Challenge 1: Encoding Tucker solutions. While the setup described above is sufficient to obtain hardness for a polynomially small ε, the encoding of the Tucker solutions

fails when one considers a constant ε. Specifically, in the computation of $L(i)$, note that most of the terms will be zero, corresponding to points that do not have label i. When K is chosen to be polynomially large, as it is in all prior works, then the values of $L(i)$ become polynomially small. This does not cause issues when ε is also polynomially small, as one can still distinguish $L(i)$ being close to zero, and $L(i)$ being far from zero. But when ε is a constant we lose that power, and the reduction breaks.

One idea is to try to get away with only a constant number K of samples and circuit copies. Indeed, prior work by Rubinstein [2018] in the context of PPAD has succeeded in performing the so-called averaging trick with only a constant number of samples. Unfortunately, there seems to be a fundamental obstacle to this kind of approach here: there are up to N stray cuts that can "destroy" up to N circuit copies, and thus any constant number K of copies will not be enough.

To address this, we define a new version of the Tucker problem where the labels have more expressive power. We call this new problem ND-StrongTucker. Briefly, this is a variant of ND-Tucker in which each point is assigned either $+1$ or -1 for *every* dimension i, and thus each point has N labels. That is, the function $\lambda : G \to \{-1, +1\}^N$ now returns a vector, such that $[\lambda(x)]_i$ tells us whether the point has label $+1$ or -1 in dimension i. The antipodality condition on the boundary can again be formulated as $\lambda(\overline{x}) = -\lambda(x)$ (for $x \in \partial G$). A solution is a set of N points $x^1, x^2, \ldots, x^N$, that are all within L_∞ distance 1 of each other, and that cover all labels, meaning that for each dimension $i \in [N]$ there exists a point x^{ℓ_1} with $[\lambda(x^{\ell_1})]_i = +1$ and a point x^{ℓ_2} with $[\lambda(x^{\ell_2})]_i = -1$. See Definition 3 for the formal definition.

Intuitively, in ND-StrongTucker the label $\lambda(x)$ carries much more information than in ND-Tucker. Indeed, in a certain sense, the label at some point x now has to pick a direction in *each* dimension $i \in [N]$, and cannot remain "neutral" in some dimension. This is exactly what our reduction to ε-Consensus-Halving requires.

We show that ND-StrongTucker is PPA-complete, even when the side-length of the grid is equal to 8 in all dimensions. We then use ND-StrongTucker in the reduction to ε-Consensus-Halving. This averts the problems mentioned above, since now each sum $L(i)$ consists of summands that are $+1$ and -1, and thus $L(i)$ will not be (constantly) close to zero unless both $+1$ and -1 appear as labels in dimension i.

To show hardness for ND-StrongTucker, we reduce from 2D-Tucker. We first show that 2D-Tucker reduces to 2D-StrongTucker by a fairly direct reduction that maps each of the labels $-2, -1, 1, 2$ from 2D-Tucker to one of the four possible vector labels in 2D-StrongTucker. Such a simple mapping is not possible in higher dimensions, however, and so we then use the hardness of 2D-StrongTucker to show hardness for ND-StrongTucker. Here we use a careful adaptation of the *snake embedding* idea that was used to reduce 2D-Tucker to ND-Tucker in Work 2. This construction allows us to decrease the width of one of the dimensions by a constant fraction, by introducing a new dimension (of small width) and folding the instance within this new dimension (see Figure 1). While this type of embedding has been used in the past, a fresh construction is needed in our case to deal with the fact that all points have N labels in an ND-StrongTucker instance.

The PPA-hard instances of 2D-Tucker have exponential width in both dimensions. Repeatedly applying the snake embedding allows us to reduce this to an instance in which all dimensions have width 8. As it turns out, in our final reduction to ε-Consensus-Halving, the constant width of the instance is not strictly necessary (an instance with polynomial widths would suffice), but we believe that the hardness for constant width may have applications elsewhere.

Challenge 2: Sampling. With this more powerful Tucker problem in hand, one could hope to obtain hardness for constant ε by simply replacing ND-Tucker by ND-StrongTucker in the reduction of Work 3. Unfortunately, there is another point in that reduction that relies on inverse polynomial ε: the sampling. Indeed, that work makes use of arithmetic gates and the so-called equi-angle sampling technique, which has been used in the past to prove PPAD-hardness for the Nash equilibrium problem [Chen et al., 2009]. Unfortunately, this sampling technique cannot be combined with constant-error-arithmetic gates. Since we have to take a polynomial number of samples (recall that the stray cuts force us to do this), and the error in each gate is constant, we will not obtain enough distinct samples to ensure that most of them are correctly decoded.

In order to overcome this obstacle, we switch to using Boolean gates, instead of arithmetic gates, like the two original works (Works 1 and 2), while keeping all the other major simplifications introduced by Work 3. Thinking in terms of Boolean gates allows us to construct a very simple, yet very powerful sampling gadget. We subdivide the input region I into subregions $I_1, \ldots, I_N$, one for each dimension. The idea is that the ith coordinate of the encoded points will be extracted from I_i. Next, we subdivide I_i into $7N$ subregions $I_{i,1}, \ldots, I_{i,7N}$. We essentially read one bit from each of those $7N$ subregions and interpret the resulting bitstring as the unary representation of a number in $[7N]$. Then, this number is scaled down to lie in $[8]$, in order to correspond to a coordinate in $G = [8]^N$. The crucial point is that we read the coordinate in unary representation and with more precision than actually needed. Thus, even if N bits fail, the final number in $[8]$ will move by at most 1.

With this simplified sampling technique in hand, it is now possible to reduce to ε-Consensus-Halving for some constant $\varepsilon > 0$.

Challenge 3: Optimizing ε. Our final challenge is to push the reduction technique introduced in Works 1 and 2, and simplified in Work 3, to its limits, by trying to obtain hardness for the largest possible value of ε. This effort results in a streamlined reduction that still follows the high-level structure presented above, but where each individual component is as lightweight as possible. Some note-worthy points are:

- Switching to Boolean gates, which was very useful to overcome the previous challenge, now becomes a necessity when one is interested in obtaining large constant values of ε. In particular, with the new sampling approach introduced above, the width of ND-StrongTucker does not limit how much we can increase ε. In other words, improving the PPA-hardness of ND-StrongTucker to grids of width less than 8 would not yield an improvement to the ε we obtain.

- Our reduction ends up only using two types of gates: NOT and NAND. Each of these two gates can be implemented by a single agent. The use of NAND instead of AND is not significant, but just for convenience (AND would require creating a NAND gate and then using a NOT gate on its output).

- The natural construction of the NAND gate requires $\varepsilon < 1/7$. In order to improve this to $\varepsilon < 1/5$, we eliminate one of the key components introduced in Work 3, the so-called constant creation region, and replace it by an ad-hoc argument which involves arguing about the parity of the number of cuts. This kind of argument is more reminiscent of Works 1 and 2.

Putting all these optimizations together, we obtain the reduction presented in Section 5, which proves PPA-hardness of ε-Consensus-Halving for any constant $\varepsilon < 1/5$. The

construction also provides a satisfying explanation for why we cannot go above 1/5 with current techniques. Indeed, it turns out that for NOT gates $\varepsilon < 1/3$ would suffice, but it is the NAND gates which require $\varepsilon < 1/5$. Other parts of the reduction would also work with $\varepsilon < 1/3$. Thus, the NAND gates are clearly identified as the bottleneck for improving ε. More generally, it can be seen that any gate that combines two bits into one in some non-trivial way (e.g., not just copying the first input bit), will require $\varepsilon < 1/5$. Nevertheless, this limitation could be lifted if the reduction was able to handle more than N stray cuts. None of the existing works provide a way to handle this, since all of them crucially rely on there being at most N stray cuts. Indeed, if there are $N+1$ stray cuts, then we can no longer argue that if a stray cut affects our circuits, then we are on the boundary of G.

4 Hardness of StrongTucker

In this section we introduce a new problem, ND-StrongTucker, and we show that it is PPA-hard for any $N \geq 2$. Our reduction from ND-StrongTucker to ε-Consensus-Halving in Section 5 will also show that the problem is in PPA, and hence the problem is PPA-complete.

4.1 Useful Terminology and Auxiliary Results

We begin by introducing some notation. Consider points $\mathbf{z}_1, \ldots, \mathbf{z}_r$ and a labelling λ, such that for any $j \in [r]$ we have $\lambda(\mathbf{z}_j) \in \{-1, +1\}^N$. We say that $\mathbf{z}_1, \ldots, \mathbf{z}_r$ *cover all labels* if for all $i \in [N]$ and $v \in \{-1, +1\}$ there exists a $j \in [r]$ with $[\lambda(\mathbf{z}_j)]_i = v$. Consider an N-dimensional grid $[m_1] \times \cdots \times [m_N]$ of points and a labelling $\lambda : [m_1] \times \cdots \times [m_N] \to L$ for some co-domain L. The *antipodal point* of a point $\mathbf{x} = (x_1, \ldots, x_N)$ that lies on the boundary of the grid (i.e., $x_i = 1$ or $x_i = m_i$ for some i) is the point $\overline{\mathbf{x}} := (m_1 - x_1 + 1, \ldots, m_N - x_N + 1)$. We say that the labelling satisfies *antipodality* if $\lambda(\overline{\mathbf{x}}) = -\lambda(\mathbf{x})$ for every $\mathbf{x}$ on the boundary.

We now present an auxiliary lemma that will be useful in this and the following section.

Lemma 4.1. *Consider $r \geq N+1$ points $\mathbf{z}_1, \ldots, \mathbf{z}_r$ and a labelling λ, such that for any $j \in [r]$ we have $\lambda(\mathbf{z}_j) \in \{-1, +1\}^N$. If these points cover all labels, then there exists a subset of these points of size at most N that covers all labels. Furthermore, we can recover these at most N points in polynomial time.*

Proof. Consider the set $T := \{\mathbf{z}_1, \ldots, \mathbf{z}_r\}$. We will first show that there exists a multiset of T with cardinality $N+1$, namely, $\mathbf{z}_1^*, \ldots, \mathbf{z}_{N+1}^*$, that covers all labels. Consider an arbitrary point from T to serve as our desired $\mathbf{z}_1^*$, without loss of generality $\mathbf{z}_1$. Its label is $\lambda(\mathbf{z}_1)$. Then, find a point $\mathbf{z}_2^* \in T$ such that $[\lambda(\mathbf{z}_2^*)]_1 = -[\lambda(\mathbf{z}_1)]_1$. Next, find a point $\mathbf{z}_3^* \in T$ such that $[\lambda(\mathbf{z}_3^*)]_2 = -[\lambda(\mathbf{z}_1)]_2$. Similarly, for $j \in \{4, 5, \ldots, N+1\}$ find a point $\mathbf{z}_j^* \in T$ such that $[\lambda(\mathbf{z}_j^*)]_{j-1} = -[\lambda(\mathbf{z}_1)]_{j-1}$. Since T covers all labels, this procedure is well-defined. The multiset $\{\mathbf{z}_1^*, \ldots, \mathbf{z}_{N+1}^*\}$ we thus obtain has cardinality $N+1$, and covers all labels.

Now consider the distinct elements of the aforementioned multiset, i.e., the set $S := \{\mathbf{z}_1^*, \ldots, \mathbf{z}_{N+1}^*\}$. If $|S| \leq N$, then we are done. It remains to handle the case where $|S| = N+1$. For the sake of contradiction, assume there is no N-subset of S that satisfies the claim of the lemma. Let us create all N-subsets of S as follows:

$$S_j = S \setminus \{\mathbf{z}_j^*\}, \quad j \in [N+1].$$

Now consider the function $f : \{S_1, \ldots, S_{N+1}\} \to [N]$ defined as:

$$f(S_j) = \min\{i \in [N] : [\lambda(\mathbf{z}_k^*)]_i = [\lambda(\mathbf{z}_\ell^*)]_i \quad \forall k, \ell \neq j\}.$$

Note that f is well-defined since, by assumption, every S_j has such a minimum index. Then, by the pigeonhole principle, this function maps two elements $S_{j'}, S_{j''}$ of its domain to the same value $i' \in [N]$, i.e. $f(S_{j'}) = f(S_{j''}) = i'$. Therefore, the set of points $S_{j'} \cup S_{j''}$ also has the property that the i'-th coordinate of all its points' labels has the same value in $\{-1, +1\}$. But by definition of the S_j's, we have $S_{j'} \cup S_{j''} = S$, thus our initial assumption that the points of S cover all labels does not hold (the label-coordinate i' is not covered), which is a contradiction.

To find the set S we need to check r many points in the worst case. To recover from S the required N-subset we need to check at most all of its $N + 1$ many N-subsets. Considering the polynomial time that the labelling circuit λ needs in order to provide us with the requested labels of the points in the above procedure, we conclude that the overall time to recover the desired N points is polynomial. □

We now formally define ND-StrongTucker.

Definition 3 (ND-StrongTucker)**.** An instance of ND-StrongTucker consists of a labelling $\lambda : [m_1] \times \cdots \times [m_N] \to \{-1, +1\}^N$ (represented by a Boolean circuit) that satisfies antipodality. A solution consists of N points[6] $\mathbf{z}_1, \ldots, \mathbf{z}_N$ that cover all labels, and such that $||\mathbf{z}_j - \mathbf{z}_k||_\infty \leq 1$ for all $j, k \in [N]$.

The following theorem states that ND-StrongTucker always has a solution.

Theorem 4.2. *Let us have an N-dimensional grid $[m_1] \times \cdots \times [m_N]$ of points and a labelling $\lambda : [m_1] \times \cdots \times [m_N] \to \{-1, +1\}^N$ that satisfies antipodality. Then, there exist N points $\mathbf{z}_1, \ldots, \mathbf{z}_N$ that cover all labels such that $||\mathbf{z}_j - \mathbf{z}_k||_\infty \leq 1$ for all $j, k \in [N]$.*

Proof. The proof of existence is indirect, and comes from the proof of PPA-inclusion of ND-StrongTucker presented in Section 5. In the aforementioned section we prove that ND-StrongTucker reduces to ε-Consensus-Halving for any constant $\varepsilon < 1/5$. And by the fact that ε-Consensus-Halving is in PPA [Filos-Ratsikas and Goldberg, 2018], we get the required inclusion. Alternatively, one could also reduce the problem to some version of Borsuk-Ulam by taking an appropriate continuous interpolation of the labelling. □

4.2 The Reduction

In this section we show the following theorem.

Theorem 4.3. ND-StrongTucker *is* PPA*-complete even when $m_i = 8$ for all $i \in [N]$.*

The remainder of this section is devoted to proving this theorem. The reduction consists of two steps. We first show that the 2D version of the problem is hard, and then we show that hardness for the 2D case implies hardness for higher dimensional instances. The following theorem shows hardness for 2D-StrongTucker via a direct reduction from 2D-Tucker.

Theorem 4.4. 2D-StrongTucker *is* PPA*-complete.*

[6] The points do not need to be distinct.

Proof. 2D-Tucker is known to be PPA-complete [Aisenberg et al., 2020]. We will reduce this problem to 2D-StrongTucker straightforwardly by just translating the labelling $\lambda_T : [m] \times [m] \to \{\pm 1, \pm 2\}$ of the former to the labelling $\lambda_{ST} : [m] \times [m] \to \{-1, +1\}^2$ of the latter as follows. For any point $\mathbf{x}$, if $\lambda_T(\mathbf{x}) = +2$ then $\lambda_{ST}(\mathbf{x}) = (+1, +1)$, if $\lambda_T(\mathbf{x}) = -2$ then $\lambda_{ST}(\mathbf{x}) = (-1, -1)$, if $\lambda_T(\mathbf{x}) = +1$ then $\lambda_{ST}(\mathbf{x}) = (+1, -1)$, and if $\lambda_T(\mathbf{x}) = -1$ then $\lambda_{ST}(\mathbf{x}) = (-1, +1)$. By definition of the problems, it is immediate that their sets of solutions are identical.

By reversing the above translation of the labelling, i.e. turning λ_{ST} to λ_T using the same mapping, we get a reduction from 2D-StrongTucker to 2D-Tucker, and hence, the former problem's membership to PPA. □

Overview of the reduction from 2D-StrongTucker to ND-StrongTucker. We will reduce 2D-StrongTucker with width $m = 2^M$ to ND-StrongTucker with width 8 for some appropriate value of $N = O(M)$. The reduction is, in essence, a careful application of the well-known snake embedding technique [Chen et al., 2009; Filos-Ratsikas and Goldberg, 2019] which was used to reduce 2D-Tucker to a Tucker problem of higher dimension. We have to carefully apply the latter technique for our problem since we need to make sure that no artificial solutions are introduced in the "folding" process, and that the folded k-dimensional instance in each step is a proper $(k+1)$-dimensional instance, meaning that it preserves antipodality. As a final step, we ensure that all dimensions have width exactly 8.

The snake embedding technique starts from the $2^M \times 2^M$ 2D-StrongTucker instance and at each step performs a "folding" on some dimension, decreasing its width to roughly 1/3 of its size, while creating a new dimension of width 8. In this way, in roughly $2 \cdot \log_3 2^M$ foldings we have created an equal amount of extra dimensions of width at most 8. In general, given a kD-StrongTucker instance for $k \geq 2$, by performing a folding on its i-th dimension, we create a $(k+1)$D-StrongTucker instance with new width $m_i' \leq \lceil \frac{m_i}{3} \rceil + 4$ and an extra $(k+1)$-st dimension of width 8. Finally, we perform two extra foldings to ensure that our initial dimensions 1 and 2 have also width 8.

Making the width of kD-StrongTucker suitable for folding. We now describe a general step of the snake embedding, that is, a step where we are given a kD-StrongTucker instance and we fold it into a $(k+1)$D-StrongTucker instance. Pick a dimension of kD-StrongTucker, without loss of generality $i \in [k]$, that has maximum width $m_i > 8$, if any. We will call this the *folding dimension*, and for some $d \in [m_i]$, let us call *d-th ray* the set of points of the grid that have coordinate d in that dimension. According to our folding procedure, the i-th dimension will have to be of width of the form $3 \cdot s + 1$, for some natural number s. Therefore, for width m_i that is not of the aforementioned size, we have to add extra copies of rays in order to bring it to the required width. When we refer to adding copies of rays we mean that we copy sets of points together with their labels. In order to preserve antipodality we have to take care of how many copies of the 1-st and m_i-th ray we will add. To achieve this, instead of adding one ray when needed, we can attach four extra rays, namely, two left of coordinate 1 and two right of coordinate m_i. Let us call the initial kD-StrongTucker instance I_{ST} and the one with proper width I_{WST}.

Let us use the following set of rules that depend on the size of m_i and preserve antipodality:

- If $m_i = 3 \cdot s' + 2$ we add one copy of the 1-st ray left of the 1-st ray and one copy of the m_i-th ray right of the m_i-th ray.

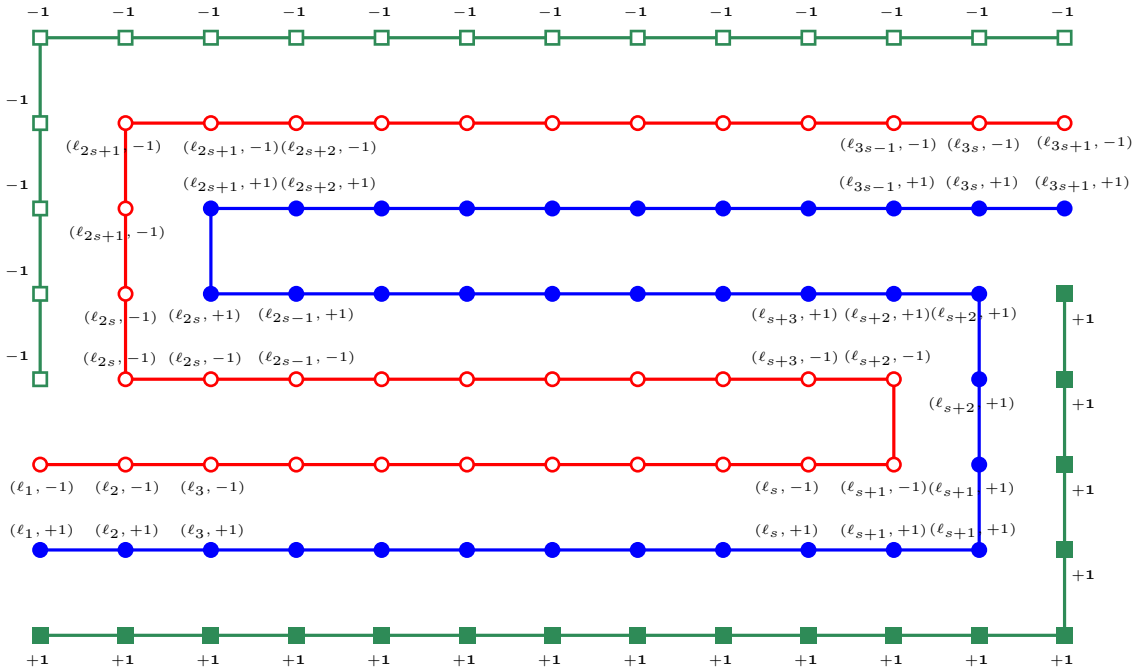

Figure 1: A slice of the $(k+1)$D-StrongTucker instance I^*_{ST} for fixed coordinates of all dimensions other than the folding dimension i (x-axis) and $k+1$ (y-axis). ℓ_j is the label λ of the point with coordinate j in dimension i in the kD-StrongTucker instance I_{WST}. By $+\mathbf{1}$ and $-\mathbf{1}$ we denote the $(k+1)$-dimensional vector (label) with all entries $+1$ and -1 respectively.

- If $m_i = 3 \cdot s' + 1$ we do not need to add any ray.
- If $m_i = 3 \cdot s'$ we add two copies of the 1-st ray left of the 1-st ray and two copies of the m_i-th ray right of the m_i-th ray.

The above additions of rays ensure that the width of the i-th dimension of I_{WST} is $3 \cdot s + 1$ for some $s \in \mathbb{N}^*$.

Copying and folding I_{WST}, and creation of the $(k+1)$D-StrongTucker instance. In essence, we create two identical (up to the turning points) copies of I_{WST} that we glue together and fold in a snake-like shape. Let us call *bottom snake* the bottom layer of I_{WST} as appears in Figure 1 (blue/shaded-circle layer), and *top snake* the top layer of I_{WST} (red/hollow-circle layer). Then we need to take care of the turns of I_{WST} so that they do not introduce artificial solutions. To achieve this, it suffices that the bottom snake is formed by copying the $(s+1)$-st and $(s+2)$-nd rays two times, and the top snake is formed by copying the $2s$-th and $(2s+1)$-st rays two times. Then, the folding in the i-th dimension of the bottom and top snakes is as demonstrated in Figure 1.

Next, we add extra rays below the bottom snake and above the top snake (green/shaded-squares and green/hollow-squares respectively in Figure 1). In particular, we add rays whose coordinates in the i-th and $(k+1)$-st dimensions are

- $(j, 1)$ for all $j \in \{1, \ldots, s+2\}$ and $(s+3, m)$ for all $m \in \{1, \ldots, 5\}$, which consist the *bottom cap*, and symmetrically,
- $(j, 8)$ for all $j \in \{2, \ldots, s+3\}$ and $(1, m)$ for all $m \in \{4, \ldots, 8\}$, which consist the *top cap*.

Let us call I^*_{ST} the resulting $(k+1)$D-StrongTucker instance. From the described folding procedure, we conclude that by folding the i-th dimension of I_{WST} for which $m_i = 3 \cdot s + 1$, we generate I^*_{ST} which has an extra $(k+1)$-st dimension of width 8 and its i-th dimension has now width $m'_i = s + 3$ (see Figure 1).

Mapping points of I^*_{ST} to points of I_{WST}. From the construction so far, we can determine a surjection of points of the bottom and top snakes in I^*_{ST} to points in I_{WST}. We only need such a surjection because, as we will show later, no point of the bottom or top cap can participate in a solution of I^*_{ST}. We will map the ray (j, m) corresponding to the coordinates of the i-th and the $(k+1)$-st dimensions of I^*_{ST} to the t-th ray corresponding to the coordinate of the i-th dimension of I_{WST}. When we say that we map ray r_1 to ray r_2 we imply that any point in r_1 with fixed coordinates in the $k-1$ of its dimensions maps to the point of r_2 with the same coordinates of these $k-1$ dimensions. The surjection is as follows.

- (j, m) for $j \in \{1, \ldots, s+1\}$ and $m \in \{2, 3\}$ maps to $t = j$.
- $(s+2, m)$ for $m \in \{2, 3\}$ maps to $t = s + 1$.
- $(s+2, m)$ for $m \in \{4, 5\}$ maps to $t = s + 2$.
- (j, m) for $j \in \{3, \ldots, s+1\}$ and $m \in \{4, 5\}$ maps to $t = 2s + 3 - j$.
- $(2, m)$ for $m \in \{4, 5\}$ maps to $t = 2s$.
- $(2, m)$ for $m \in \{6, 7\}$ maps to $t = 2s + 1$.
- (j, m) for $j \in \{3, \ldots, s+3\}$ and $m \in \{6, 7\}$ maps to $t = 2s - 2 + j$.

Labelling I^*_{ST}. Having specified the structure of the $(k+1)$D-StrongTucker instance, we have to determine the labels of its points. By the construction described in the previous paragraph, the added rays determine the first k label-coordinates of the points in the bottom and top snakes. The $(k+1)$-st label-coordinate of each point in the two snakes is determined as follows: for the bottom snake its value is $+1$ and for the top snake its value is -1. Finally, for all points of the bottom cap the label is $+\mathbf{1}$, i.e., all label-coordinates get value $+1$, and similarly, for all points of the top cap the label is $-\mathbf{1}$.

Correctness of the reduction. So far we have made sure that at each step of the folding procedure the k-dimensional instance I_{ST} at hand and also its modified version I_{WST} will be proper kD-StrongTucker instances. Now we will prove correctness of the reduction by showing that every solution of the final ND-StrongTucker instance corresponds to a solution in the initial 2D-StrongTucker instance. We will show this by proving that at every step $k \geq 2$ of the folding procedure, every solution of the $(k+1)$D-StrongTucker instance I^*_{ST} corresponds to a solution of the kD-StrongTucker instance I_{WST}.

Suppose that $S' = \{\mathbf{z}'_1, \ldots, \mathbf{z}'_{k+1}\}$ is a solution to I^*_{ST}. Let us prove the following claim.

Claim 1. *No point of S' belongs to the bottom or top cap.*

Proof. Let us first consider the bottom cap. For the sake of contradiction, suppose that a point of S' belongs to the bottom cap. Then, since $||\mathbf{z}'_\ell - \mathbf{z}'_m||_\infty \leq 1$ for all $\ell, m \in [k+1]$, all other points belong to the union of the bottom snake and the bottom cap. But then, the $(k+1)$-st label-coordinate of all points in S' is $+1$, contradicting the property of S' that its points cover all labels.

Similarly, if one of the points from S' belonged to the top cap, then the labels of all points in S' would have their $(k+1)$-st coordinate equal to -1, a contradiction. □

By the labelling in the folding we have specified earlier, any solution S' has to include at least one point of the bottom snake and at least one point of the top snake, otherwise their $(k+1)$-st label-coordinates would be the same - either $+1$ or -1 - contradicting the property of covering all labels. Let us call a point $\mathbf{z}'$ of I^*_{ST} a *bottom corner point* if it belongs to one of the copied rays of the bottom snake. Similarly, let us call it a *top corner point* if it belongs to one of the copied rays in the top snake.

Consider for each $\mathbf{z}'_j \in S'$ the point $\mathbf{z}^*_j$ of I_{WST} to which it is mapped according to the respective paragraph above. Let $S^* = \{\mathbf{z}^*_1, \ldots, \mathbf{z}^*_{k+1}\}$ be the set of the corresponding k-dimensional points of I_{WST}.

Let the labellings of I^*_{ST} and I_{WST} be denoted by λ' and λ, respectively. According to the mapping of points of the former to points of the latter instance we have defined in the respective paragraph above, it is immediate that if we have $||\mathbf{z}'_\ell - \mathbf{z}'_m||_\infty \leq 1$ for every $\mathbf{z}'_\ell, \mathbf{z}'_m \in S'$, then $||\mathbf{z}^*_\ell - \mathbf{z}^*_m||_\infty \leq 1$ for every $\mathbf{z}^*_\ell, \mathbf{z}^*_m \in S^*$. Furthermore, the labellings that we have defined above copy the labellings of the respective points that we map from I^*_{ST}. Therefore, if S' covers all labels in λ' (which has $k+1$ coordinates) then S^* covers all labels in λ (which has k coordinates). Observe that $|S^*| \leq k+1$. If $|S^*| = k+1$, from Lemma 4.1 we get that there is a k-subset of S^* which is a solution to I_{WST}. Furthermore, this solution can be found in polynomial time from S^*.

Putting everything together. We are now ready to finish the proof of Theorem 4.3. By repetitions of the cycle $I_{ST} \to I_{WST} \to I^*_{ST}$, we fold the widest dimension of the kD-StrongTucker starting from $k = 2$ until all dimensions' widths are at most 8. As we showed above, this is guaranteed to happen after linearly many foldings, by the fact that every folding dimension i has width $m_i = 3 \cdot s + 1 > 8$ for some $s \in \mathbb{N}^*$, and after the folding its width is $m'_i = s + 3$, while a $(k+1)$-st dimension of width 8 has been created (see Figure 1).

We will now show the final step that makes all the dimensions' widths exactly 8. Recall that we have started from the $2^M \times 2^M$ 2D-StrongTucker instance, and by repeatedly folding dimensions 1 and 2 we have generated extra dimensions of width 8. Therefore, the only dimensions we need to take care of are the aforementioned two. Let us consider dimension $i \in \{1, 2\}$ and recall that after a folding, the folded dimension's width is $s + 3$ for some $s \in \mathbb{N}^*$, while initially it was $3 \cdot s + 1$. Let us denote by m^t_i the width of the dimension after t foldings. As mentioned earlier, our folding technique reduces the size of the folding dimension i from m^{t-1}_i to $m^t_i \leq \left\lceil \frac{m^{t-1}_i}{3} \right\rceil + 4 \leq \frac{m^{t-1}_i}{3} + 5$, for any $t \in \mathbb{N}^*$. This recursion induces the following inequality:

$$m^t_i \leq \frac{m^0_i}{3^t} + \frac{15}{2} \cdot \left(1 - \frac{1}{3^t}\right).$$

Therefore, we can ensure that the left-hand side is at most 8 by forcing the right-hand side to be at most 8, which can be achieved in at most $\lceil \log_3(2 \cdot m^0_i - 15) \rceil$ steps. Before the first folding (i.e. after making the width proper for folding), the width of dimension $i \in \{1, 2\}$ of our initial 2D-StrongTucker instance will be $m^0_i \leq 2^M + 4$, therefore after some number $t^* \leq \lceil \log_3(2^{M+1} - 7) \rceil$ of foldings we have $m^{t^*}_i \leq 8$, at which point we stop.

If $m^{t^*}_i = 8$ then we are done. If $m^{t^*}_i < 8$, notice that it will necessarily be an even number. That is because, for any given $t \in \mathbb{N}^*$, if $m^{t-1}_i = 3 \cdot s + 1$ is even then $m^t_i = s + 3$ is even, and we have started with $m^0_i = 2^M + 2 \cdot p$ for some $p \in \{0, 1, 2\}$ (to make sure

that it is of the form $3 \cdot s + 1$ before the folding, as described in the respective paragraph above). Since $m_i^{t^*}$ is even, let us copy its 1-st ray $(16 - m_i^{t^*})/2$ times to the left and its $m_i^{t^*}$-th ray $(16 - m_i^{t^*})/2$ times to the right, and create a modified instance with $m_i' = 16$ by following the procedure described in the respective paragraph above. Recall that this procedure ensures that the modified instance preserves antipodality. Now perform a final folding which will bring the width from $16 = 3 \cdot 5 + 1$ to $5 + 3 = 8$.

Finally, inclusion of ND-StrongTucker in PPA comes from the reduction of ND-StrongTucker to ε-Consensus-Halving for any constant $\varepsilon < 1/5$ presented in Section 5. As shown by Filos-Ratsikas and Goldberg [2018], ε-Consensus-Halving is in PPA, which implies the required inclusion.

5 Main Reduction

In this section, we prove our main result, Theorem 1.1. Namely, for any constant $\varepsilon < 1/5$, we present a polynomial-time reduction from ND-StrongTucker to ε-Consensus-Halving. In Section 5.4 we explain how our reduction can be modified to work with 3-block uniform valuations, thus proving Theorem 1.2.

Fix any $\varepsilon \in [0, 1/5)$. Let λ be an instance of ND-StrongTucker, i.e., $\lambda : [8]^N \to \{-1, +1\}^N$ is provided as a Boolean circuit. We use $\text{size}(\lambda)$ to denote the representation size of the Boolean circuit λ. Note that, in particular, $\text{size}(\lambda) \geq N$. We show how to construct an instance $\text{CH}_\varepsilon(\lambda)$ of ε-Consensus-Halving in time polynomial in $\text{size}(\lambda)$, such that from any solution of $\text{CH}_\varepsilon(\lambda)$ we can extract in polynomial time a solution to λ.

5.1 Pre-processing

Construction of the modified circuit $\widehat{\lambda}$. The first step of the reduction is to construct a slightly modified version of λ, which will be more convenient to work with. First of all, we will not think of bits as lying in $\{0, 1\}$, but, instead, in $\{-1, +1\}$. Here, -1 will represent bit 0 ("False"), and $+1$ will represent bit 1 ("True").

With this interpretation in mind, the modified circuit, which we denote by $\widehat{\lambda}$, is defined as follows. The input to $\widehat{\lambda}$ consists of $7N^2$ bits, that we think of as a matrix $x \in \{-1, +1\}^{N \times 7N}$. We use $x_{i,j} \in \{-1, +1\}$ to denote the (i, j) entry, and $x_i \in \{-1, +1\}^{7N}$ to denote the ith row. The circuit outputs N bits representing a label $\{-1, +1\}^N$. On input x, the circuit performs the following computations.

1. For each $i \in [N]$, compute

$$\phi_i(x) := \left\lceil \frac{8N + 1/2 + \sum_{j=1}^{7N} x_{i,j}}{2N} \right\rceil \in [8]. \tag{1}$$

2. Compute and output $\lambda(\phi(x)) \in \{-1, +1\}^N$.

In time polynomial in $\text{size}(\lambda)$ we construct a Boolean circuit $\widehat{\lambda}$ that performs these computations, and only uses NOT gates and NAND gates. Note that other logical gates can easily be simulated using these two gates.

Intuitively, the circuit $\widehat{\lambda}$ does the following. For any $i \in [N]$, $x_i \in \{-1, +1\}^{7N}$ is interpreted as representing a number between 1 and 8 with precision roughly $1/N$ (in unary representation). That number is then rounded to obtain an integer $\phi_i(x) \in [8]$. Why do we use more bits than needed to represent a number in [8]? The reason is that this representation is robust to flipping a few bits. Indeed, it is easy to check that flipping up

to N bits of $x_i \in \{-1,+1\}^{7N}$ changes the value of $\phi_i(x)$ by at most 1. As a result, we obtain the following:

Claim 2. *If $x, x' \in \{-1,+1\}^{N\times 7N}$ are such that for all $i \in [N]$, x_i and x'_i differ in at most N bits, then $\|\phi(x) - \phi(x')\|_\infty \le 1$.*

Proof. If x_i and x'_i differ in at most N bits, then

$$\left|\sum_{j=1}^{7N} x_{i,j} - \sum_{j=1}^{7N} x'_{i,j}\right| \le 2N$$

and as a result $|\phi_i(x) - \phi_i(x')| \le 1$ by Equation (1). □

The circuit $\widehat{\lambda}$ consists of m gates $g_1, \ldots, g_m$, where $m \ge N$ and $m \le \text{size}(\widehat{\lambda}) \le \text{poly}(\text{size}(\lambda))$. For each $t \in [m]$, $g_t = (g_{t_1}, g_{t_2}, T)$, where $t_1, t_2 \in [t-1] \cup ([N] \times [7N])$ are the inputs to the gate, and $T \in \{\text{NOT}, \text{NAND}\}$ indicates the type of gate. Note that an input g_{t_1} to a gate g_t can be of two types: when $t_1 \in [t-1]$, then g_{t_1} is simply another ("earlier") gate of the circuit; when $t_1 \in [N] \times [7N]$, then $g_{t_1} = g_{(i,j)}$, which we interpret as the (i,j)th input to the circuit, i.e., $x_{i,j}$. Note that when $T = \text{NOT}$, the second input g_{t_2} is ignored. The output of the circuit $\widehat{\lambda}$ is given by the last N gates, i.e., $g_{m-N+1}, \ldots, g_m$.

5.2 Construction of the Instance

We now begin with the description of the ε-Consensus-Halving instance $\text{CH}_\varepsilon(\lambda)$ that we construct. Instead of working with the interval $[0,1]$, we will describe the construction on an interval $R = [0, \text{poly}(\text{size}(\lambda))]$. The valuations of the agents can then easily be scaled down to $[0,1]$.

Input and circuit regions. The interval R is subdivided into two subintervals: interval I on the left, and interval C on the right. Interval I is called the "Input region", while C is called the "Circuit region". The interval I is further subdivided into intervals $I_1, I_2, \ldots, I_N$ from left to right. Next, each interval I_i is subdivided into intervals $I_{i,1}, \ldots, I_{i,7N}$. Finally, each interval $I_{i,j}$ is subdivided into intervals $I^1_{i,j}, \ldots, I^{3N}_{i,j}$. Each of those final small intervals has length 1, i.e., $|I^k_{i,j}| = 1$. Thus, the total length of interval I is $N \cdot 7N \cdot 3N = 21N^3$.

The interval C is subdivided into intervals $C^1, C^2, \ldots, C^{3N}$. We think of each C^k as being associated to a separate "copy" of the circuit $\widehat{\lambda}$. Next, each interval C^k is subdivided into intervals $C^k_1, \ldots, C^k_m$, where we recall that m is the number of gates of $\widehat{\lambda}$. Finally, each interval C^k_t is subdivided into intervals $C^k_{t,\ell}, C^k_{t,c}, C^k_{t,r}, C^k_{t,a}$. The intervals $C^k_{t,\ell}, C^k_{t,c}, C^k_{t,r}, C^k_{t,a}$ have length 1 each, and are called the *left/center/right/auxiliary* subinterval of C^k_t, respectively. Putting everything together, we see that $|C| = 3N \cdot m \cdot 4 = 12mN$ and thus $|R| = |I \cup C| = 21N^3 + 12mN = \text{poly}(\text{size}(\lambda))$.

Agents. The instance $\text{CH}_\varepsilon(\lambda)$ will have exactly $n = 3N \cdot m \cdot 2 + N$ agents. Namely, for each $k \in [3N]$ and $t \in [m]$, there is a *gate agent* α^k_t and an *auxiliary agent* β^k_t. We think of these agents as "belonging" to the interval C^k_t. Furthermore, there are also *feedback agents* $\gamma_1, \ldots, \gamma_N$. We will define the valuation functions for all these agents below, but first we have to introduce the notion of the value encoded by an interval.

Value of an interval. Consider any solution S of our instance $\mathrm{CH}_\varepsilon(\lambda)$. Then $S = (R^+, R^-)$ is a partition of R into two parts R^+ and R^- using at most n cuts. Without loss of generality, we can assume that S has the following property: the right-most end of R lies in R^+. Indeed, if this is not the case, then swapping R^+ and R^- yields a solution that satisfies this.

In any solution $S = (R^+, R^-)$, we can assign a value in $[-1, 1]$ to any interval $J \subset R$, $|J| = 1$, in a natural way:

$$\mathsf{val}_S(J) := \mu(J \cap R^+) - \mu(J \cap R^-)$$

where μ is the Lebesgue measure on $\mathbb{R}$. When it is clear from the context, we will omit the subscript S. We say that the value of an interval J is *pure*, if $\mathsf{val}(J) \in \{-1, +1\}$, i.e., it can be interpreted as a bit.

For $k \in [3N]$, $i \in [N]$, and $j \in [7N]$, we let

$$x^k_{i,j} := \mathsf{val}(I^k_{i,j}) \in [-1, 1].$$

Furthermore, for $k \in [3N]$ and $t \in [m]$, we let

$$g^k_t := \mathsf{val}(C^k_{t,c}) \in [-1, 1].$$

For convenience, we also define $g^k_t := x^k_{i,j}$, when $t = (i, j) \in [N] \times [7N]$, i.e., when t refers to an input of the circuit, and not a gate.

We think of $x^1, \ldots, x^{3N}$ as $3N$ possible inputs to our circuit $\widehat{\lambda}$. Of course, $\widehat{\lambda}(x^k)$ is only well-defined if x^k is pure, i.e., if $x^k \in \{-1, +1\}^{N \times 7N}$. We can make the following crucial observations.

Claim 3. *In any solution S where at most N cuts lie in the interior of interval I, it holds that, if x^{k_1} and x^{k_2} are both pure, then $\|\phi(x^{k_1}) - \phi(x^{k_2})\|_\infty \leq 1$ and $\lambda(\phi(x^{k_i})) = \widehat{\lambda}(x^{k_i})$ for $i = 1, 2$.*

Proof. The statement $\lambda(\phi(x^{k_i})) = \widehat{\lambda}(x^{k_i})$ follows by the construction of $\widehat{\lambda}$. It remains to prove that $\|\phi(x^{k_1}) - \phi(x^{k_2})\|_\infty \leq 1$. Since the interior of I contains at most N cuts, it follows that for each $i \in [N]$, the interior of the interval I_i contains at most N cuts. As a result, there exists a subset $P_i \subseteq [7N]$ with $|P_i| \geq 7N - N = 6N$ such that for all $j \in P_i$ the interior of interval $I_{i,j}$ does not contain any cuts. This means that for all $j \in P_i$, the intervals $I^{k_1}_{i,j}$ and $I^{k_2}_{i,j}$ have the same value, i.e., $x^{k_1}_{i,j} = \mathsf{val}(I^{k_1}_{i,j}) = \mathsf{val}(I^{k_2}_{i,j}) = x^{k_2}_{i,j}$. Thus, since $|P_i| \geq 6N$, $x^{k_1}_i$ and $x^{k_2}_i$ differ in at most N bits. Since this holds for all $i \in [N]$, the claim follows by Claim 2. □

Claim 4. *In any solution S where at most $N - 1$ cuts lie in the interior of interval I, it holds that, if x^k is pure, then $\widehat{\lambda}(-x^k) = -\widehat{\lambda}(x^k)$.*

Proof. Since the interior of I contains at most $N-1$ cuts, there exists $s \in [N]$ such that the interior of I_s does not contain any cuts. As a result, $x^k_{s,j_1} = \mathsf{val}(I^k_{s,j_1}) = \mathsf{val}(I^k_{s,j_2}) = x^k_{s,j_2}$ for all $j_1, j_2 \in [7N]$. By the definition of ϕ (Equation (1)), it follows that $\phi_s(x^k) \in \{1, 8\}$. Thus, by the boundary conditions of λ, we obtain that $\lambda(\overline{\phi(x^k)}) = -\lambda(\phi(x^k))$, where $\overline{\phi_i(x^k)} = 9 - \phi_i(x^k)$ for all $i \in [N]$. Since $\lambda(\phi(x^k)) = \widehat{\lambda}(x^k)$, it remains to show that $\phi(-x^k) = \overline{\phi(x^k)}$.

Fix any $i \in [N]$ and consider $\phi_i(x^k) = q \in [8]$. By the definition of ϕ (Equation (1)), it follows that

$$(q-1) \cdot 2N < 8N + 1/2 + \sum_{j=1}^{7N} x^k_{i,j} \leq q \cdot 2N$$

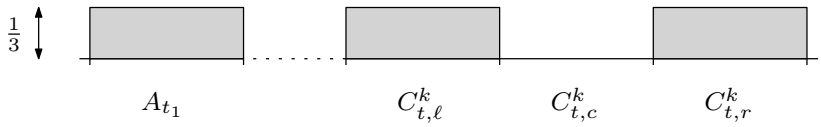

Figure 2: The density function of the valuation of an agent α^k_t implementing a NOT-gate.

which implies that

$$(8-q)\cdot 2N+1 \leq 8N+1/2-\sum_{j=1}^{7N} x^k_{i,j} < (9-q)\cdot 2N+1$$

and finally

$$(8-q)\cdot 2N < 8N+1/2-\sum_{j=1}^{7N} x^k_{i,j} \leq (9-q)\cdot 2N.$$

But, by the definition of ϕ (Equation (1)), this exactly means that $\phi_i(-x^k) = 9-q = 9-\phi_i(x^k) = \overline{\phi_i(x^k)}$. □

Auxiliary agents. For $k \in [3N]$ and $t \in [m]$, the auxiliary agent β^k_t has a very simple valuation function $v_{\beta^k_t}$: the density function of the valuation has value 1 in $C^k_{t,a}$, and value 0 everywhere else. This corresponds to having a block of volume 1 lying in interval $C^k_{t,a}$. We immediately obtain the following observation.

Claim 5. *For all $k \in [3N]$ and $t \in [m]$ there must be a cut in the interior of $C^k_{t,a}$.*

Proof. If there is no cut in the interior of $C^k_{t,a}$, then $|v_{\beta^k_t}(R^+) - v_{\beta^k_t}(R^-)| = 1 > \varepsilon$. □

Gate agents: NOT. Let $t \in [m]$ be such that $g_t = (g_{t_1}, g_{t_2}, \text{NOT})$. Then, for any $k \in [3N]$, the goal of gate agent α^k_t is to enforce the corresponding gate constraint, namely $g^k_t = \mathsf{val}(C^k_{t,c}) = \text{NOT}(g^k_{t_1}) = -g^k_{t_1}$. The density function of the valuation $v_{\alpha^k_t}$ is constructed as follows: it has value $1/3$ in $C^k_{t,\ell} \cup C^k_{t,r} \cup A_{t_1}$, and value 0 everywhere else. Here A_{t_1} is defined as

- $A_{t_1} = C^k_{t_1,c}$, when $t_1 \in [t-1]$,
- $A_{t_1} = I^k_{i,j}$, when $t_1 = (i,j) \in [N] \times [7N]$.

Note that $\mathsf{val}(A_{t_1}) = g^k_{t_1}$. See Figure 2 for an illustration of the gate.

Claim 6. *For all $t \in [m]$ such that $g_t = (g_{t_1}, g_{t_2}, \text{NOT})$, and all $k \in [3N]$, it holds that:*

- *there must be a cut in the interior of $C^k_{t,\ell} \cup C^k_{t,c} \cup C^k_{t,r}$;*
- *if there is exactly one cut in the interior of $C^k_{t,\ell} \cup C^k_{t,c} \cup C^k_{t,r}$, and the input to the gate is pure (namely, $g^k_{t_1} \in \{-1,+1\}$), then the output is pure (i.e., $g^k_t \in \{-1,+1\}$), and $g^k_t = \text{NOT}(g^k_{t_1})$.*

Proof. Assume, towards a contradiction, that the interior of $C^k_{t,\ell} \cup C^k_{t,c} \cup C^k_{t,r}$ does not contain any cuts. Then, in particular, $C^k_{t,\ell}$ and $C^k_{t,r}$ are both contained in R^+ or both contained in R^-. This implies that $|v_{\alpha^k_t}(R^+) - v_{\alpha^k_t}(R^-)| \geq 2/3 - 1/3 = 1/3 > \varepsilon$, a contradiction.

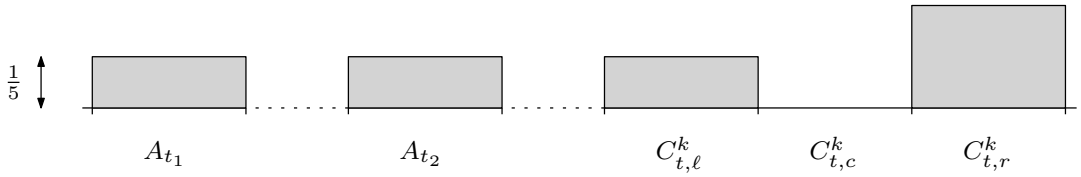

Figure 3: The density function of the valuation of an agent α^k_t implementing a NAND-gate.

Now consider the case where the interior of $C^k_{t,\ell} \cup C^k_{t,c} \cup C^k_{t,r}$ contains exactly one cut. Let A_{t_1} be as defined above. Recall that $g^k_{t_1} = \mathsf{val}(A_{t_1})$ and $g^k_t = \mathsf{val}(C^k_{t,c})$. If $\mathsf{val}(A_{t_1}) = +1$, then it cannot be that $\mathsf{val}(C^k_{t,c}) \neq -1$. Indeed, since the interior of $C^k_{t,\ell} \cup C^k_{t,c} \cup C^k_{t,r}$ contains a single cut, $\mathsf{val}(C^k_{t,c}) \neq -1$ implies that at least one of $C^k_{t,\ell}$ or $C^k_{t,r}$ is contained in R^+. But since A_{t_1} is also contained in R^+, this implies $|v_{\alpha^k_t}(R^+) - v_{\alpha^k_t}(R^-)| \geq 2/3 - 1/3 = 1/3 > \varepsilon$, a contradiction. Thus, it must be that $\mathsf{val}(C^k_{t,c}) = -1$. Similarly, we can show that $\mathsf{val}(A_{t_1}) = -1$ implies $\mathsf{val}(C^k_{t,c}) = +1$. □

Gate agents: NAND. Let $t \in [m]$ be such that $g_t = (g_{t_1}, g_{t_2}, \text{NAND})$. Then, for any $k \in [3N]$, the goal of gate agent α^k_t is to enforce the corresponding gate constraint, namely $g^k_t = \mathsf{val}(C^k_{t,c}) = \text{NAND}(g^k_{t_1}, g^k_{t_2}) = -(g^k_{t_1} \wedge g^k_{t_2})$. The density function of the valuation $v_{\alpha^k_t}$ is constructed as follows: it has value $2/5$ in $C^k_{t,r}$, value $1/5$ in $A_{t_1} \cup A_{t_2} \cup C^k_{t,\ell}$, and value 0 everywhere else. The intervals A_{t_1}, A_{t_2} are defined as above in the description of the NOT-gate. See Figure 3 for an illustration of the gate.

Claim 7. *For all $t \in [m]$ such that $g_t = (g_{t_1}, g_{t_2}, \text{NAND})$, and all $k \in [3N]$, it holds that:*

- *there must be a cut in the interior of $C^k_{t,\ell} \cup C^k_{t,c} \cup C^k_{t,r}$;*
- *if there is exactly one cut in the interior of $C^k_{t,\ell} \cup C^k_{t,c} \cup C^k_{t,r}$, and the inputs to the gate are pure (namely, $g^k_{t_1}, g^k_{t_2} \in \{-1, +1\}$), then the output is pure (i.e., $g^k_t \in \{-1, +1\}$), and*
  - *if the left end of C^k_t lies in R^+, then $g^k_t = \text{NAND}(g^k_{t_1}, g^k_{t_2})$;*
  - *if the left end of C^k_t lies in R^-, then $g^k_t = -\text{NAND}(-g^k_{t_1}, -g^k_{t_2})$.*

Proof. Assume, towards a contradiction, that the interior of $C^k_{t,\ell} \cup C^k_{t,c} \cup C^k_{t,r}$ does not contain any cuts. Then, $C^k_{t,\ell}$ and $C^k_{t,r}$ are both contained in R^+ or both contained in R^-. This implies that $|v_{\alpha^k_t}(R^+) - v_{\alpha^k_t}(R^-)| \geq 3/5 - 2/5 = 1/5 > \varepsilon$, a contradiction.

Now consider the case where the interior of $C^k_{t,\ell} \cup C^k_{t,c} \cup C^k_{t,r}$ contains exactly one cut. Recall that $g^k_{t_1} = \mathsf{val}(A_{t_1})$, $g^k_{t_2} = \mathsf{val}(A_{t_2})$, and $g^k_t = \mathsf{val}(C^k_{t,c})$. First, assume that the left end of C^k_t lies in R^+. If $\mathsf{val}(A_{t_1}) = \mathsf{val}(A_{t_2}) = +1$, then the cut must lie in $C^k_{t,\ell}$. Otherwise, $C^k_{t,\ell}$ lies in R^+, just like A_{t_1} and A_{t_2}, which implies $|v_{\alpha^k_t}(R^+) - v_{\alpha^k_t}(R^-)| \geq 3/5 - 2/5 = 1/5 > \varepsilon$, a contradiction. Since the cut lies in $C^k_{t,\ell}$, it follows that $C^k_{t,c}$ lies in R^-, i.e., $\mathsf{val}(C^k_{t,c}) = -1$, as desired.

If $\mathsf{val}(A_{t_1}) = -1$, then the cut must lie in $C^k_{t,r}$. Indeed, otherwise, $C^k_{t,r}$ lies in R^-, just like A_{t_1}, which implies $|v_{\alpha^k_t}(R^+) - v_{\alpha^k_t}(R^-)| \geq 3/5 - 2/5 = 1/5 > \varepsilon$, a contradiction. Since the cut lies in $C^k_{t,r}$, it follows that $C^k_{t,c}$ lies in R^+, i.e., $\mathsf{val}(C^k_{t,c}) = +1$, as desired. The exact same analysis also applies to the case where $\mathsf{val}(A_{t_2}) = -1$ instead. Thus, we obtain $\mathsf{val}(C^k_{t,c}) = \text{NAND}(\mathsf{val}(A_{t_1}), \mathsf{val}(A_{t_2}))$.

It remains to consider the setting where the left end of C_t^k lies in R^-, instead of R^+. The same type of case analysis applied to this setting yields $\mathsf{val}(C_{t,c}^k) = -\mathrm{NAND}(-\mathsf{val}(A_{t_1}), -\mathsf{val}(A_{t_2}))$. □

Remark 1. Note that the proof of Claim 7 crucially made use of the fact that $\varepsilon < 1/5$. In fact, it turns out that this is the only point in the reduction where this is needed. The rest of the reduction can be made to work for any $\varepsilon < 1/3$. In particular, it is not hard to see that the proof of Claim 6 only made use of the assumption $\varepsilon < 1/3$.

Feedback agents. For $i \in [N]$, feedback agent γ_i has the following valuation function v_{γ_i}: the density function of v_{γ_i} has value $1/3N$ over $\cup_{k=1}^{3N} C_{m-N+i,c}^k$, and value 0 everywhere else. Recall that the interval $C_{m-N+i,c}^k$ corresponds to the gate g_{m-N+i} of $\widehat{\lambda}$, which is the ith output of $\widehat{\lambda}$. For every $k \in [3N]$, define $y^k \in [-1,1]^N$ by letting

$$y_i^k := g_{m-N+i}^k = \mathsf{val}(C_{m-N+i,c}^k)$$

for all $i \in [N]$. Intuitively, y^k corresponds to the output of the kth circuit region C^k. By construction of γ_i, we immediately obtain:

Claim 8. *For all $i \in [N]$, it holds that*

$$\left| \frac{1}{3N} \sum_{k=1}^{3N} y_i^k \right| \leq \varepsilon.$$

Proof. We can write

$$\begin{aligned} \left| v_{\gamma_i}(R^+) - v_{\gamma_i}(R^-) \right| &= \left| \sum_{k=1}^{3N} v_{\gamma_i}(C_{m-N+i,c}^k \cap R^+) - v_{\gamma_i}(C_{m-N+i,c}^k \cap R^-) \right| \\ &= \left| \frac{1}{3N} \sum_{k=1}^{3N} \mathsf{val}(C_{m-N+i,c}^k) \right| = \left| \frac{1}{3N} \sum_{k=1}^{3N} y_i^k \right| \end{aligned}$$

and at any solution we must have $|v_{\gamma_i}(R^+) - v_{\gamma_i}(R^-)| \leq \varepsilon$. □

We have now completed the construction of the instance $\mathrm{CH}_\varepsilon(\lambda)$. It is easy to check that this construction can be performed in time polynomial in $\mathrm{size}(\lambda)$.

5.3 Correctness of the Reduction

It remains to prove the correctness of the reduction, namely, that from any solution $S = (R^+, R^-)$ to $\mathrm{CH}_\varepsilon(\lambda)$ we can extract a solution to the ND-StrongTucker instance λ. We show this by presenting and proving a sequence of claims.

Claim 9. *For every $k \in [3N]$, the interior of C^k contains at least $2m$ cuts.*

Proof. This immediately follows from Claim 5, Claim 6 and Claim 7, by observing that every gate agent and auxiliary agent forces a cut to lie in the interior of some interval, and all these intervals are pairwise disjoint. □

Intuitively, a circuit region C^k will correctly perform computations as long as it does not contain more than $2m$ cuts (and thus, by Claim 9 above, exactly $2m$ cuts). Furthermore, for the computations to be meaningful, the inputs to the circuit, namely $x^k_{i,j} = \mathsf{val}(I^k_{i,j})$, should also be pure. This motivates defining the "good" copies of the circuit as

$$G := \left\{ k \in [3N] : \text{the interior of } \overline{C}^k := C^k \cup \left(\bigcup_{(i,j)\in[N]\times[7N]} I^k_{i,j} \right) \text{ contains at most } 2m \text{ cuts} \right\}.$$

Claim 10. *It holds that $|G| \geq 2N$.*

Proof. Note, first of all, that for $k_1 \neq k_2$, the interior of $\overline{C}^{k_1}$ is disjoint from the interior of $\overline{C}^{k_2}$. Furthermore, by Claim 9 we know that, for each $k \in [3N]$, the interior of C_k contains at least $2m$ cuts. Since there are $n = 3N \cdot 2m + N$ agents, and thus also at most that many cuts, it follows that there remain at most N "free" cuts. As a result, the number of $\overline{C}^k$ that contain more than $2m$ cuts can be at most N. □

Claim 11. *For all $k \in G$, we have that $x^k \in \{-1,+1\}^{N\times 7N}$, and*

- *if the left end of C^k lies in R^+, then $y^k = \widehat{\lambda}(x^k)$;*
- *if the left end of C^k lies in R^-, then $y^k = -\widehat{\lambda}(-x^k)$.*

Proof. Since $k \in G$, by definition of G and by Claim 9, no cut lies in the interior of $I^k_{i,j}$ for all $(i,j) \in [N] \times [7N]$. As a result, $x^k_{i,j} = \mathsf{val}(I^k_{i,j}) \in \{-1,+1\}$, i.e., x^k is pure.

Consider the case where the left end of C^k lies in R^+. By definition of G and by Claim 9 it follows that the interior of C^k contains exactly $2m$ cuts, and for each $t \in [m]$, the interior of C^k_t contains exactly two cuts, namely one in $C^k_{t,\ell} \cup C^k_{t,c} \cup C^k_{t,r}$, and one in $C^k_{t,a}$. As a result, it holds that for each $t \in [m]$, the left end of C^k_t lies in R^+. Thus, since x^k is pure, it follows by Claim 6 and Claim 7 that the output of the kth copy of the first gate is pure, i.e., $g^k_1 \in \{-1,+1\}$, and that the value of the gate is computed correctly. By induction, it follows that $g^k_t \in \{-1,+1\}$ for all $t \in [m]$, and that all the gates are computed correctly. In particular, we obtain that $y^k = \widehat{\lambda}(x^k)$.

Now, consider the case where the left end of C^k lies in R^-. By the same argument as above, it follows that for each $t \in [m]$, the left end of C^k_t lies in R^-. As above, since x^k is pure, and by Claim 6 and Claim 7, we obtain that the kth copy of the first gate $g_1 = (g_{t_1}, g_{t_2}, T)$ is pure, i.e., $g^k_1 \in \{-1,+1\}$, and

- if $T = \text{NOT}$: $g^k_1 = \text{NOT}(g^k_{t_1}) = -\text{NOT}(-g^k_{t_1})$;
- if $T = \text{NAND}$: $g^k_1 = -\text{NAND}(-g^k_{t_1}, -g^k_{t_1})$.

By induction, it follows that $g^k_t \in \{-1,+1\}$ for all $t \in [m]$, and that $g^k_t = -g_t[-x^k]$, i.e., each gate has the opposite value from the one it would have if the input to the circuit was $-x^k$. In particular, we obtain that $y^k = -\widehat{\lambda}(-x^k)$. □

We are now ready to complete the proof. Putting everything together, we can prove a stronger version of Claim 11.

Claim 12. *For all $k \in G$, we have that $x^k \in \{-1,+1\}^{N\times 7N}$, and $y^k = \widehat{\lambda}(x^k)$.*

Proof. In order to prove the claim, we consider two distinct cases. First, let us assume that the interior of I contains at least N cuts. Recall that the number of agents is $n = 3N \cdot 2m + N$, and thus the total number of cuts is at most $3N \cdot 2m + N$. Since the interior of I contains at least N cuts, and, for each $k \in [3N]$, the interior of C^k contains at least $2m$ cuts (Claim 9), it follows that the interior of I contains exactly N cuts, and, for each $k \in [3N]$, the interior of C^k contains exactly $2m$ cuts. As a result, for each $k \in [3N]$, the left end of C^k lies in R^+, because the number of cuts in C^k is even (using the fact that without loss of generality the right end of R lies in R^+). By Claim 11, it follows that for each $k \in G$, $x^k \in \{-1, +1\}^{N \times 7N}$ and $y^k = \widehat{\lambda}(x^k)$.

Now, consider the second case, namely that the interior of I contains at most $N-1$ cuts. By Claim 11 we know that for all $k \in G$, $x^k \in \{-1, +1\}^{N \times 7N}$ and $y^k \in \{\widehat{\lambda}(x^k), -\widehat{\lambda}(-x^k)\}$. However, since the interior of I contains at most $N-1$ cuts, it follows by Claim 4 that $\widehat{\lambda}(x^k) = -\widehat{\lambda}(-x^k)$. Thus, for all $k \in G$, it holds that $y^k = \widehat{\lambda}(x^k)$. □

Claim 13. *The set of points $\{\phi(x^k) : k \in G\}$ yields a solution to the N*D-StrongTucker *instance λ.*

Proof. By Claim 3 and Claim 12, we know that for all $k \in G$, $x^k \in \{-1, +1\}^{N \times 7N}$ and $y^k = \widehat{\lambda}(x^k) = \lambda(\phi(x^k))$. Furthermore, for all $k_1, k_2 \in G$, we have $\|\phi(x^{k_1}) - \phi(x^{k_2})\|_\infty \leq 1$. Thus, it remains to show that the points in $\{x^k : k \in G\}$ cover all the labels of $\widehat{\lambda}$. Towards a contradiction, assume that this is not the case. Then, there exists $i \in [N]$ and $b \in \{-1, +1\}$ such that $y_i^k = b$ for all $k \in G$. But then, since $|G| \geq 2N$ (Claim 10), and $|y_i^k| \leq 1$ for all $k \in [3N]$,

$$\left| \frac{1}{3N} \sum_{k=1}^{3N} y_i^k \right| \geq \frac{1}{3N}(|G| - (3N - |G|)) \geq 1/3 > \varepsilon$$

which contradicts Claim 8, namely, the feedback agent γ_i cannot be satisfied in that case. It follows that the points in $\{x^k : k \in G\}$ do indeed cover all the labels of $\widehat{\lambda}$. As a result, we can extract a solution to λ from $\{\phi(x^k) : k \in G\}$ by using Lemma 4.1. □

Finally, note that, given a solution S of $\mathrm{CH}_\varepsilon(\lambda)$, we can in polynomial time compute G, then $\{\phi(x^k) : k \in G\}$, and finally use Lemma 4.1 to extract N points that are a solution to λ. This completes the proof of correctness for the reduction.

5.4 Extension to 3-Block Uniform Valuations

The proof that we presented above can be modified to prove Theorem 1.2, namely that the result holds even if we restrict the valuations to be 3-block uniform. Recall that an agent has a 3-block uniform valuation function if the density function of the valuation is non-zero in at most three intervals, and in each such interval it has the same non-zero value. Filos-Ratsikas et al. [2023] have proved that the problem remains PPA-complete even for 2-block uniform valuations, but their hardness result only holds for polynomially small ε.

The following modifications to the proof of Theorem 1.1 are needed to obtain Theorem 1.2:

- **Number of copies:** Instead of $3N$ copies of the circuit, we use $20N$ copies of the circuit. In particular, every interval $I_{i,j}$ is now subdivided into intervals $I_{i,j}^1, \ldots, I_{i,j}^{20N}$.
- **NOT-gates:** Auxiliary agents and NOT-gate agents already have 3-block uniform valuations. Thus, no change is needed there.

- **NAND-gates:** Unfortunately, NAND-gate agents are not 3-block uniform. To address this, we modify each NAND-gate agent as follows: the density function of the valuation has value $1/5$ in each of the three intervals A_{t_1}, A_{t_2}, and $C^k_{t,\ell} \cup C^k_{t,c} \cup C^k_{t,r}$, and value 0 everywhere else. This valuation function is now 3-block uniform, but the value of the output of the gate is no longer encoded in the standard way. Indeed, letting J_t denote the interval of length $\delta := (1/5 - \varepsilon)/2 > 0$ centered around the point $C^k_{t,\ell} \cap C^k_{t,c}$, we can prove an analogue of Claim 7, where the output value is no longer stored in $C^k_{t,c}$, but in J_t instead (i.e., $J_t \subseteq R^+$ means that the output is $+1$, and $J_t \subseteq R^-$ means that the output is -1). As a result, some additional modifications are needed to correctly read the output of such a gate.

  Without loss of generality, we can assume that in the circuit $\widehat{\lambda}$, every NAND-gate is always followed by a NOT-gate, i.e., the output of a NAND-gate can only be used as an input to a NOT-gate (and, in particular, can also not be an output of the circuit). This can easily be ensured by introducing two consecutive NOT-gates wherever that is needed. With this in hand, for each NOT-gate $g_{t'}$ that takes as input the output of a NAND-gate g_t, we will modify the corresponding NOT-gate agent as follows: the density function of the valuation has a block of length δ and height $1/3\delta$ in J_t, and two more such blocks, one in each of $C^k_{t',\ell}$ and $C^k_{t',r}$ (anywhere inside those intervals). We can prove an analogue of Claim 6 to show that this agent behaves as a standard NOT-gate, except that the input is read from J_t instead of $C^k_{t,c}$. See Figure 4 for an illustration of the modified NAND-gate, together with its corresponding modified NOT-gate.

- **Feedback agents:** The last remaining agents that do not have 3-block uniform valuations are the feedback agents $\gamma_1, \ldots, \gamma_N$. To address this we modify the feedback mechanism as follows. We add an "Output region" O between the input region I and the circuit region C. Interval O is subdivided into intervals $O_1, \ldots, O_n$, and each O_i is subdivided into intervals $O^1_i, \ldots, O^{20N}_i$. Finally, each O^k_i is subdivided into three intervals of length 1: $O^k_{i,\ell}$, $O^k_{i,c}$, and $O^k_{i,r}$. For each $i \in [N]$ and $k \in [20N]$, using an additional NOT-gate based in O^k_i, we copy the value of the ith output of the kth copy of the circuit into the interval $O^k_{i,c}$. Finally, we define the feedback agent γ_i as follows: the density function is uniform over O_i, i.e., it has value $1/60N$ over O_i, and value 0 elsewhere.

  With this modified construction we can then prove an analogue of Claim 13. The main observation is that if, say, $y^k_i = +1$ for all $k \in G$, then $\mu(O_i \cap R^+) \geq 2|G|$ and $\mu(O_i \cap R^-) \leq |G| + 3(20N - |G|)$. Using the fact that $|G| \geq 20N - N$, it follows that agent γ_i is not satisfied, since

$$\frac{1}{60N}(\mu(O_i \cap R^+) - \mu(O_i \cap R^-)) \geq \frac{1}{60N}(4|G| - 60N) \geq \frac{80N - 4N - 60N}{60N} = \frac{4}{15} > \varepsilon.$$

  Note that here we crucially used the fact that we now have $20N$ copies instead of just $3N$.

6 Conclusion

So far, ε-Consensus-Halving has been the starting point for every PPA-hardness result for problems that do not include a circuit in their definition. We have resolved the complexity of the problem for constant approximations by showing that it is PPA-complete

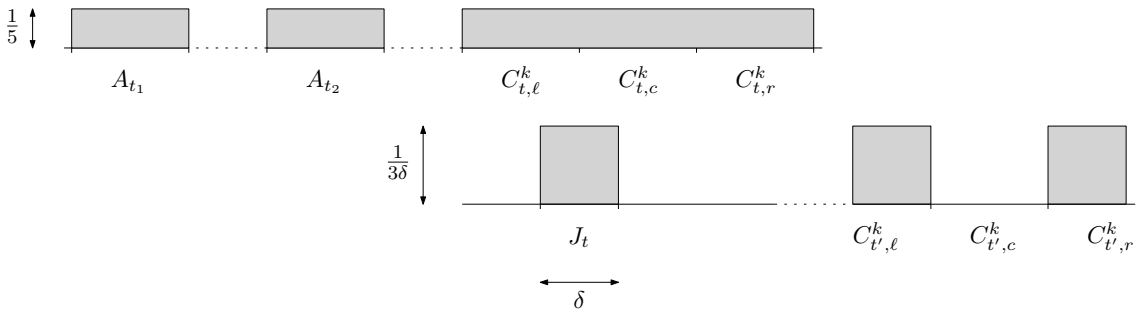

Figure 4: The valuation function of an agent implementing a modified NAND-gate (top), and the valuation function of an agent implementing a modified NOT-gate reading the output of the NAND-gate (bottom).

for every constant $\varepsilon < 1/5$. We expect that this will be very useful to obtain further strong inapproximability results for PPA problems.

There are several remaining questions related to Consensus-Halving.

- **Improve ε beyond 1/5.** The current bottleneck of our technique is the NAND gate. We conjecture that the ε we derive for a NAND gate implemented via a single agent, or even a constant number of agents, is optimal. Hence, we believe that a new technique would be needed to get PPA-hardness for a larger ε.

  The NAND gate, and, in fact, any gadget taking two bits as input and having a non-trivial output bit (e.g., not just copying the first input bit), has to balance out two constraints. Let I denote the subinterval of the consensus-halving interval that encodes the input(s) of the gate, and O the subinterval that encodes the output. Then the two constraints are the following.

  1. The agent encoding the gate has to have enough value in I (with respect to ε), so that "what happens in I has some effect on the agent".
  2. The agent encoding the gate has to have enough value in O (with respect to ε), so that there is necessarily a cut in O. This is to avoid extra stray cuts, which the current reduction framework (like all previous ones) cannot handle.

  Together with the fact that the agent's valuation has to be normalized to 1, these two constraints yield that $\varepsilon < 1/5$ for a Boolean gate with two inputs, and $\varepsilon < 1/3$ for a Boolean gate with one input.

- **Prove an upper bound.** So far, no algorithm is known for solving ε-Consensus-Halving (with n cuts) for some constant $\varepsilon < 1$, even for piecewise constant valuations with positive value on two intervals only. The only known upper bounds are for very special cases [Deligkas et al., 2022b; Filos-Ratsikas et al., 2023] or with additional cuts [Alon and Graur, 2021].

- **Constant number of agents.** For a constant number of agents with explicitly represented piecewise constant valuations (as modelled in this paper), ε-Consensus-Halving can be solved in polynomial time by a simple enumeration algorithm [Filos-Ratsikas et al., 2023]. But what is the complexity of the problem if we are not given the whole valuations upfront, but instead can only efficiently evaluate them? In [Deligkas et al., 2022c] it was proven that in this setting the problem is PPA-complete for 3 agents with *non-additive* valuations. It seems that

proving hardness for the more standard additive valuation model would require radically new ideas.

- **Necklaces with few beads per colour.** What is the computational complexity of NecklaceSplitting with a constant number of beads per colour? Our hardness result, which directly uses the reduction presented by Filos-Ratsikas and Goldberg [2018], constructs necklaces with polynomially-many beads for each colour. There appears to be no straightforward way to reduce this number to a constant. On the other hand, to the best of our knowledge, there is no efficient algorithm that solves NecklaceSplitting when every colour appears at most four times.

Acknowledgements

The first author was supported by EPSRC grant EP/X039862/1 "NAfANE: New Approaches for Approximate Nash Equilibria". The second author was supported by EPSRC grant EP/W014750/1 "New Techniques for Resolving Boundary Problems in Total Search". The work was completed while the fourth author was affiliated with the group of Operations Research at the Technical University of Munich. This author's work was supported by the Alexander von Humboldt Foundation with funds from the German Federal Ministry of Education and Research (BMBF).

References

Ron Aharoni, Noga Alon, Eli Berger, Maria Chudnovsky, Dani Kotlar, Martin Loebl, and Ran Ziv. Fair representation by independent sets. In *A Journey Through Discrete Mathematics*, pages 31–58. Springer, 2017. doi:10.1007/978-3-319-44479-6_2.

James Aisenberg, Maria Luisa Bonet, and Sam Buss. 2-D Tucker is PPA complete. *Journal of Computer and System Sciences*, 108:92–103, 2020. doi:10.1016/j.jcss.2019.09.002.

Meysam Alishahi and Frédéric Meunier. Fair splitting of colored paths. *The Electronic Journal of Combinatorics*, 24(3):P3.41, 2017. doi:10.37236/7079.

Noga Alon. Splitting necklaces. *Advances in Mathematics*, 63(3):247–253, 1987. doi:10.1016/0001-8708(87)90055-7.

Noga Alon and Andrei Graur. Efficient splitting of necklaces. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 14:1–14:17, 2021. doi:10.4230/LIPIcs.ICALP.2021.14.

Noga Alon and Douglas B. West. The Borsuk-Ulam Theorem and Bisection of Necklaces. *Proceedings of the American Mathematical Society*, 98(4):623–628, 1986. doi:10.2307/2045739.

Luis Barba, Alexander Pilz, and Patrick Schnider. Sharing a pizza: bisecting masses with two cuts. *arXiv preprint*, abs/1904.02502, 2019. URL https://arxiv.org/abs/1904.02502.

Eleni Batziou, Kristoffer Arnsfelt Hansen, and Kasper Høgh. Strong approximate Consensus Halving and the Borsuk-Ulam theorem. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 24:1–24:20, 2021. doi:10.4230/LIPIcs.ICALP.2021.24.

Alexander Black, Umur Cetin, Florian Frick, Alexander Pacun, and Linus Setiabrata. Fair splittings by independent sets in sparse graphs. *Israel Journal of Mathematics*, 236: 603–627, 2020. doi:10.1007/s11856-020-1980-5.

Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM*, 56(3):14:1–14:57, 2009. doi:10.1145/1516512.1516516.

Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009. doi:10.1137/070699652.

Argyrios Deligkas, John Fearnley, Themistoklis Melissourgos, and Paul G. Spirakis. Computing exact solutions of consensus halving and the Borsuk-Ulam theorem. *Journal of Computer and System Sciences*, 117:75–98, 2021. doi:10.1016/j.jcss.2020.10.006.

Argyrios Deligkas, John Fearnley, and Themistoklis Melissourgos. Pizza sharing is PPA-hard. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI)*, pages 4957–4965, 2022a. doi:10.1609/aaai.v36i5.20426.

Argyrios Deligkas, John Fearnley, Themistoklis Melissourgos, and Paul G. Spirakis. Approximating the existential theory of the reals. *Journal of Computer and System Sciences*, 125:106–128, 2022b. doi:10.1016/j.jcss.2021.11.002.

Argyrios Deligkas, Aris Filos-Ratsikas, and Alexandros Hollender. Two's company, three's a crowd: Consensus-halving for a constant number of agents. *Artificial Intelligence*, 313, 2022c. doi:10.1016/j.artint.2022.103784. 103784.

Argyrios Deligkas, John Fearnley, Alexandros Hollender, and Themistoklis Melissourgos. Pure-circuit: Tight inapproximability for PPAD. *Journal of the ACM*, 71(5):31:1–31:48, 2024. doi:10.1145/3678166.

Xiaotie Deng, Zhe Feng, and Rucha Kulkarni. Octahedral Tucker is PPA-complete. Technical Report TR17-118, Electronic Colloquium on Computational Complexity (ECCC), 2017. URL `https://eccc.weizmann.ac.il/report/2017/118/`.

Kousha Etessami and Mihalis Yannakakis. On the complexity of Nash equilibria and other fixed points. *SIAM Journal on Computing*, 39(6):2531–2597, 2010. doi:10.1137/080720826.

Aris Filos-Ratsikas and Paul W. Goldberg. Consensus halving is PPA-complete. In *Proceedings of the 50th ACM Symposium on Theory of Computing (STOC)*, pages 51–64, 2018. doi:10.1145/3188745.3188880.

Aris Filos-Ratsikas and Paul W. Goldberg. The complexity of splitting necklaces and bisecting ham sandwiches. In *Proceedings of the 51st ACM Symposium on Theory of Computing (STOC)*, pages 638–649, 2019. doi:10.1145/3313276.3316334.

Aris Filos-Ratsikas and Paul W. Goldberg. The complexity of Necklace Splitting, Consensus-Halving, and Discrete Ham Sandwich. *SIAM Journal on Computing*, 52(2):STOC19–200–STOC19–268, 2023. doi:10.1137/20M1312678.

Aris Filos-Ratsikas, Søren Kristoffer Still Frederiksen, Paul W. Goldberg, and Jie Zhang. Hardness results for Consensus-Halving. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 24:1–24:16, 2018. doi:10.4230/LIPIcs.MFCS.2018.24.

Aris Filos-Ratsikas, Alexandros Hollender, Katerina Sotiraki, and Manolis Zampetakis. A topological characterization of modulo-p arguments and implications for necklace splitting. In *Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2615–2634, 2021. doi:10.1137/1.9781611976465.155.

Aris Filos-Ratsikas, Alexandros Hollender, Katerina Sotiraki, and Manolis Zampetakis. Consensus-Halving: Does it ever get easier? *SIAM Journal on Computing*, 52(2):412–451, 2023. doi:10.1137/20m1387493.

Charles H. Goldberg and Douglas B. West. Bisection of Circle Colorings. *SIAM Journal on Algebraic Discrete Methods*, 6(1):93–106, 1985. doi:10.1137/0606010.

Paul W. Goldberg, Alexandros Hollender, Ayumi Igarashi, Pasin Manurangsi, and Warut Suksompong. Consensus halving for sets of items. *Mathematics of Operations Research*, 47(4):3357–3379, 2022. doi:10.1287/moor.2021.1249.

Ishay Haviv. The complexity of finding fair independent sets in cycles. In *Proceedings of the 12th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 4:1–4:14, 2021. doi:10.4230/LIPIcs.ITCS.2021.4.

Ishay Haviv. The complexity of finding fair independent sets in cycles. *computational complexity*, 31(2), 2022. doi:10.1007/s00037-022-00233-6.

Charles R. Hobby and John R. Rice. A moment problem in L1 approximation. *Proceedings of the American Mathematical Society*, 16(4):665–670, 1965. doi:10.2307/2033900.

Alexandros Hollender. The classes PPA-k: Existence from arguments modulo k. *Theoretical Computer Science*, 885:15–29, 2021. doi:10.1016/j.tcs.2021.06.016.

Alfredo Hubard and Roman Karasev. Bisecting measures with hyperplane arrangements. *Mathematical Proceedings of the Cambridge Philosophical Society*, 169(3):639–647, 2020. doi:10.1017/S0305004119000380.

Roman N. Karasev, Edgardo Roldán-Pensado, and Pablo Soberón. Measure partitions using hyperplanes with fixed directions. *Israel Journal of Mathematics*, 212:705–728, 2016. doi:10.1007/s11856-016-1303-z.

Nimrod Megiddo and Christos H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991. doi:10.1016/0304-3975(91)90200-L.

Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994. doi:10.1016/S0022-0000(05)80063-7.

Aviad Rubinstein. Inapproximability of Nash equilibrium. *SIAM Journal on Computing*, 47(3):917–959, 2018. doi:10.1137/15M1039274.

Patrick Schnider. The complexity of sharing a pizza. *Computing in Geometry and Topology*, 1(1):4:1–4:19, 2022. doi:10.57717/CGT.V1I1.5.

Forest W. Simmons and Francis E. Su. Consensus-halving via theorems of Borsuk-Ulam and Tucker. *Mathematical social sciences*, 45(1):15–25, 2003. doi:10.1016/S0165-4896(02)00087-2.

Albert W. Tucker. Some topological properties of disk and sphere. In *Proceedings of the First Canadian Math. Congress, Montreal*, pages 286–309. University of Toronto Press, 1945.

A Additional Cuts

In this section, we prove Theorem 1.3, which we restate here for convenience.

Theorem 1.3. *ε-Consensus-Halving is PPA-complete for all $\varepsilon < 1/5$, even if all agents have 3-block uniform valuations, and even if $n + n^{1-\delta}$ cuts are allowed for some constant $\delta \in (0, 1]$, where n is the number of agents.*

Proof. We use the same approach used to prove Corollary 3.2 in [Filos-Ratsikas et al., 2023]. Let $\varepsilon < 1/5$ and $\delta \in (0, 1]$. Consider an instance CH of ε-Consensus-Halving where all agents have 3-block uniform valuations and let n denote the number of agents. By Theorem 1.2 we know that finding a solution in such instances using at most n cuts is PPA-hard. We show how to reduce this to the problem of finding a solution in an instance with N agents (who all still have 3-block uniform valuations), but using up to $N + N^{1-\delta}$ cuts.

We construct an instance CH$'$ that is composed of c completely disjoint copies of the instance CH. This means that the instance is defined on the interval $[0, c]$, with the ith copy living in subinterval $[i-1, i]$. (It is easy to renormalize the instance CH$'$ to be defined on interval $[0, 1]$ without affecting any of the arguments.) The number of copies c will be fixed below, but importantly it must be polynomial in the size of the original instance CH. The number of agents in the new instance is $N = c \cdot n$.

Consider any solution of the new instance CH$'$ that uses at most $N+(c-1) = c\cdot n+(c-1)$ cuts. By the pigeonhole principle, at least one of the copies will be cut by at most n cuts. Since the copies are disjoint, this will yield a solution to the original instance CH.

It remains to show that we can pick a number of copies c that is polynomial in the size of the original instance CH, while also ensuring that $N + (c-1) \geq N + N^{1-\delta}$. Letting $c := n^k$ and recalling that $N = c \cdot n$, this inequality can be rewritten as $n^k - 1 \geq n^{(k+1)(1-\delta)}$. Now since $\delta > 0$, there exists a sufficiently large k (depending only on δ) such that the inequality is satisfied for sufficiently large n. □