SIAM J. COMPUT.
Vol. 52, No. 2, pp. STOC19-200–STOC19-268




# THE COMPLEXITY OF NECKLACE SPLITTING, CONSENSUS-HALVING, AND DISCRETE HAM SANDWICH*

ARIS FILOS-RATSIKAS† AND PAUL W. GOLDBERG‡

**Abstract.** We resolve the computational complexity of three problems known as NECKLACE SPLITTING, CONSENSUS-HALVING, and DISCRETE HAM SANDWICH, showing that they are PPA-complete. For NECKLACE SPLITTING, this result is specific to the important special case in which two thieves share the necklace. These are the first PPA-completeness results for problems whose definition does not contain an explicit circuit, thus settling the status of PPA as a class that captures the complexity of such "natural' problems.

**Key words.** TFNP, NECKLACE-SPLITTING, COMPUTATIONAL COMPLEXITY, PPA

**AMS subject classifications.** 68Q15, 68Q17, 68R05

**DOI.** 10.1137/20M1312678

## 1. Introduction.

The complexity class TFNP was defined by Megiddo and Papadimitriou [65] and contains *total search problems* in NP, i.e., problems that always have polynomial-time checkable solutions. However, the class does not seem to have complete problems and, moreover, no problem in TFNP can be NP-complete unless NP=co-NP [64]. Consequently, alternative notions of computational hardness need to be developed and applied in our effort to understand the many and varied problems in TFNP that seem to be intractable.

The complexity class PLS defined by Johnson, Papadimitriou, and Yannakakis [55] and the classes PPAD, PPA, and PPP defined by Papadimitriou [71] are subclasses of TFNP associated with various combinatorial principles that guarantee totality. Each principle has a corresponding definition of a computational problem whose totality applies that principle in the most general way possible and a complexity class of problems reducible to it. In more detail,

- PLS consists of problems whose totality invokes the principle that every directed acyclic graph has a sink vertex;
- PPAD consists of problems whose totality is based on the principle that given a source in a directed graph whose vertices have in-degree and out-degree at most 1, there exists another degree-1 vertex;
- PPA differs from PPAD in that the graph need not be directed; being a more general principle, PPA is thus a superset of PPAD;
- PPP, based on the pigeonhole principle, consists of problems reducible to PIGEONHOLE CIRCUIT.

Of these complexity classes, only PLS, PPAD, and PPP have succeeded in capturing the complexity of "natural" computational problems. PLS corresponds with a wide range of local optimization problems [55], including searching for pure equilibria of congestion games [1]. PPAD corresponds most prominently with Nash equilibrium computation in general games [19, 24]. Recently, Sotiraki, Zampetakis, and Zirdelis

*Received by the editors January 16, 2020; accepted for publication (in revised form) October 12, 2021; published electronically February 28, 2022. The present paper is based on two conference papers [37, 38].

https://doi.org/10.1137/20M1312678

†Department of Computer Science, University of Liverpool, Liverpool, UK (aris.filos-ratsikas@liverpool.ac.uk, https://www.arisfilosratsikas.com).

‡Department of Computer Science, University of Oxford, Oxford, UK (paul.goldberg@cs.ox.ac.uk, http://www.cs.ox.ac.uk/people/paul.goldberg/index1.html).

[76] identified the first natural problem which is complete for PPP. Prior to this work, natural PPA-complete problems had not been proven to exist, where "natural" here has the very specific meaning of problems that do not explicitly contain a circuit in their definition.[1] The importance of identifying such problems was brought up as early as Papadimitriou's original paper [71] and was later reiterated in several works [49, 18, 24, 30].

The contributions of the present paper are twofold:

- We identify the first examples of natural PPA-complete problems, namely, the Necklace-Splitting problem with two thieves, the approximate Consensus-Halving problem, and the Discrete Ham Sandwich problem, dispelling the suspicion that such problems might not exist.
- These problems have been of separate interest from the complexity landscape of TFNP. As discussed in section 1.2, besides being explicitly raised as open problems in [71], the complexity of Necklace-Splitting and Discrete Ham Sandwich has been studied in works dating back to the 1980s.

Below, we provide an overview of the computational classes PPA and PPAD, as well as the three aforementioned problems and the related work on them.

**1.1. The classes PPA and PPAD.** The complexity classes PPA and PPAD were introduced by Papadimitriou [71] in 1994, in an attempt to classify various natural problems in the class TFNP [65]. PPA is the class of problems reducible to Leaf (Definition 1.1), and a PPA-complete problem is one that is polynomial-time equivalent to Leaf.

Definition 1.1 (Leaf). *An instance of the problem* Leaf *consists of an undirected graph $G$ whose vertices have degree at most 2; $G$ has $2^n$ vertices represented by bitstrings of length $n$; $G$ is presented concisely via a circuit that takes as input a vertex and outputs its neighbour(s). We stipulate that vertex $0^n$ has degree 1. The challenge is to find some other vertex having degree 1.*

PPAD has a similar definition, but the (implicit) graph is directed and we are given a vertex of outdegree 1 (a source) and we are looking for another vertex of indegree 1 (a sink) or outdegree 1 (another source); the associated canonical problem for PPAD is called End of Line.

Over the years, various important problems were proven to be complete for the class PPAD, such as the complexity of many versions of Nash equilibrium [24, 19, 34, 66, 73, 20], market equilibrium computation [23, 17, 79, 21, 74], and others [32, 57, 45]. On the other hand, the problems that are known to be PPA-complete are rather limited. Some examples include the following. Aisenberg, Bonet, and Buss [2] introduce the problem 2D-Tucker: suppose we have a coloring of an exponentially fine grid on a square region, the coloring being concisely represented via a circuit. Tucker's lemma [78] (the discrete version of the well-known Borsuk–Ulam theorem [15]) guarantees that if certain boundary conditions are obeyed, then two adjacent[2] squares in the grid will get opposite colors. 2D-Tucker is the search for such a solution, or alternatively a violation of the boundary conditions. As it happens, we use 2D-Tucker as the starting-point for our reductions here. Deng et al. [30] showed PPA-completeness

[1] The term "natural" as used here is attributed to Grigni [49]. A more intuitive definition of natural problems is of those that have been defined outside of the context of the subclasses of TFNP; the problems that we show to be PPA-complete in this paper are natural according to both definitions.

[2] Throughout this paper, two regions are considered to be adjacent whenever they meet at a single point; for example, they do not have to share a facet.

for finding fully coloured points of triangulations of various nonoriented surfaces; the colorings are presented concisely via a circuit. Recently, Deng, Feng, and Kulkarni [31] showed that *Octahedral Tucker* is PPA-complete, reducing from 2D-TUCKER and using a snake-embedding style technique that packages-up the exponential grid in two dimensions, into a grid of constant size in high dimension. Belovs et al. [10] show PPA-completeness for novel problems presented in terms of arithmetic circuits representing instances of the Chevalley–Warning theorem and Alon's Combinatorial Nullstellensatz. A common characteristic of all of these problems is that they are not natural in the sense of our earlier criterion, and as noted above, the existence of such problems was hitherto an open question.

With regard to the connection with topological fixpoint search problems, the distinction between PPAD and PPA seems to revolve around whether we are searching for a fixpoint in an oriented topological space or an unoriented one. For example, while Papadimitriou [71] showed that it's PPAD-complete to find a Sperner solution in a three-dimensional cube, Grigni [49] showed that it's PPA-complete to find a solution to Sperner's lemma in a 3-manifold consisting of the product of a Möbius strip and a line segment. The two-dimensional versions of these results are given in [18, 30]. Goldberg and Hollender [45] show that the search problem corresponding to the *hairy ball theorem*—where the space is oriented—is PPAD-complete.

Finally, note that PPA-completeness is at least as strong a notion of computational hardness as PPAD-completeness, corresponding with PPAD being a subset of PPA. Despite the apparent similarity between their definitions, there is more progress in basing the hardness of PPA on standard cryptographic assumptions: FACTORING can be reduced to PPA (with a randomized reduction) [54], while so far, the hardness of PPAD has relied on problems from indistinguishability obfuscation [12, 42] or from a variant of the Fiat–Shamir transformation [22]. Garg, Pandey, and Srinivasan [43] make progress in weakening the cryptographic assumptions on which to base the hardness of PPAD, but these are still less satisfying than in the case of PPA.

**1.2. The computational problems.** Below we provide details and definitions for the computational problems that we consider in this paper, as well as informal statements of our main results.

**1.2.1. Necklace-Splitting.** We start with the definition of the problem.

DEFINITION 1.2 (Necklace-Splitting). *In the* $k$-NECKLACE-SPLITTING *problem there is an open necklace with* $ka_i$ *beads of color* $i$ *for* $1 \leq i \leq n$. *An "open necklace" means that the beads form a string, not a cycle. The task is to cut the necklace in* $(k-1) \cdot n$ *places and partition the resulting substrings into* $k$ *collections, each containing precisely* $a_i$ *beads of color* $i$, $1 \leq i \leq n$.

In Definition 1.2, $k$ is thought of as the number of thieves who desire to split the necklace in such a way that the beads of each color are equally shared. In this paper, usually we have $k = 2$ and we refer to this special case as NECKLACE-SPLITTING.

The *Necklace-Splitting problem* was introduced in a 1982 paper of Bhatt and Leiserson [11, section 5], where it arose in the context of VLSI circuit design (the version defined in [11] is the 2-thief case proved PPA-complete in the present paper). In 1985 and 1986, the 2-thief case was shown to have guaranteed solutions (as defined in Definition 1.2) by Goldberg and West [44] and Alon and West [6] and then in 1987, Alon [3] proved existence of solutions for $k$ thieves as well. Early papers that explicitly raise its complexity-theoretic status as an open problem are [44] and [4, 5]. Subsequently, the Necklace-Splitting problem was found to be closely related to "paint-shop

scheduling," a line of work in which several papers such as [67, 70, 69] explicitly mention the question of the computational complexity of Necklace-Splitting. Meunier [67] notes that the search for a *minimum* number of cuts admitting a fair division (which may be smaller than the number $(k-1)n$ that is guaranteed to suffice) is NP-hard, even for a subclass of instances of the 2-thief case. (That is a result of Bonsma, Epping, and Hochstättler [14] for the "paint shop problem with words," equivalent to 2-thief Necklace-Splitting with two beads of each color.)

The problem for $k = 2$ was known to be in PPA from [71]; here we prove that it is PPA-complete. The corresponding theorem is the following.

Theorem 1.3. Necklace-Splitting *is* PPA*-complete when there are* $k = 2$ *thieves.*

Theorem 1.3 gives a convincing negative answer to Meunier and Neveu's questions [69] about possible polynomial-time solvability or membership of PPAD for Necklace-Splitting; likewise it runs counter to Alon's cautious optimism at ICM 1990 ([5, section 4]) that the problem may be solvable in polynomial time.

If we knew that $k$-Necklace-Splitting belonged to PPA for other values of $k$, we could of course make the blanket statement "Necklace-Splitting is PPA-complete." For numbers of $k$ which are powers of 2, we can extend our PPA-membership result; we provide more details in subsection 9.1. For more general values of $k$, however, the proofs that Necklace-Splitting is a total search problem for $k > 2$ [3, 68] do *not* seem to boil down to the parity argument on an undirected graph, and thus the class PPA does not seem to be the right candidate for the problem. Interestingly, Papadimitriou [71] (implicitly) also defined a number of computational complexity classes related to PPA, coined PPA-$p$, where $p$ denotes a prime power which is at least 2 (with PPA-2=PPA). Following the conference versions of the papers associated with this paper, Filos-Ratsikas et al. [40] proved that $k$-Necklace-Splitting is in PPA-$p$, leaving the PPA-$p$-completeness of the problem as an important open question. We discuss these classes further in subsection 9.1 as well.

**1.2.2. Consensus-Halving.** The Consensus-Halving problem involves a set of $n$ agents each of whom has a valuation function on a one-dimensional line segment $A$ (the "cake," in cake-cutting parlance). Consider the problem of selecting $k$ "cut points" in $A$ that partition $A$ into $k+1$ pieces, then labelling each piece either "positive" or "negative" in such a way that each agent values the positive pieces equally to the negative ones. In 2003, Simmons and Su [75] showed that this can always be done for $k = n$; their proof applies the Borsuk–Ulam theorem and is a proof of existence analogous to Nash's famous existence proof of equilibrium points of games, proved using Brouwer's or Kakutani's fixed point theorem. Significantly, Borsuk–Ulam is the *undirected* version of Brouwer, and already from [71] we know that it relates to PPA, making Consensus-Halving a candidate for PPA-completeness. As detailed in Definition 1.4 below, we assume that valuations are presented as step functions using the logarithmic cost model of numbers.

Definition 1.4 ($\varepsilon$-Consensus-Halving [75]). *An instance $I_{CH}$ incorporates, for $1 \le i \le n$, a nonnegative measure $\mu_i$ of a finite line interval $A = [0, x]$, where each $\mu_i$ integrates to 1 and $x > 0$ is part of the input. We assume that $\mu_i$ are step functions represented in a standard way, in terms of the endpoints of intervals where $\mu_i$ is constant, and the value taken in each such interval. We use the bit model (logarithmic cost model) of numbers. $I_{CH}$ specifies a value $\varepsilon \ge 0$ also using the bit model. We regard $\mu_i$ as the value function held by agent $i$ for subintervals of $A$.*

*A solution consists first of a set of* $n$ cut points *in* $A$ *(also given in the bit model of numbers). These points partition* $A$ *into (at most)* $n+1$ *subintervals, and the second element of a solution is that each subinterval is labelled* $A_+$ *or* $A_-$*. This labelling is a correct solution provided that for each* $i$, $|\mu_i(A_+) - \mu_i(A_-)| \leq \varepsilon$*, i.e., each agent has a value in the range* $[\frac{1}{2} - \frac{\varepsilon}{2}, \frac{1}{2} + \frac{\varepsilon}{2}]$ *for the subintervals labelled* $A_+$ *(hence also values the subintervals labelled* $A_-$ *in that range).*

We assume without loss of generality that in a valid solution, labels $A_+$ and $A_-$ alternate. We also assume that the alternating label sequence begins with label $A_+$ on the left-hand side (LHS) of $A$ (i.e., $A_+$ denotes the leftmost label in a Consensus-Halving solution). The Consensus-Halving problem of Definition 1.4 is a computational version of the *Hobby–Rice theorem* [51].

Most of the effort in this paper is devoted to proving that $\varepsilon$-Consensus-Halving is PPA-hard, when $\varepsilon$ is an inversely polynomial function of the number of agents $n$; we note that membership in PPA was already known from [37]. We have the following theorem.

Theorem 1.5. $\varepsilon$-Consensus-Halving *is* PPA*-complete for some inverse polynomial* $\varepsilon$*.*

With this at hand, we can actually reduce $\varepsilon$-Consensus-Halving to Necklace-Splitting, to prove the PPA-completeness of the later problem. In fact, in section 8, we obtain a computational equivalence between generalizations of the two problems. This kind of relation should not be surprising, seeing as most of the existence proofs for the Necklace-Splitting problem go via a continuous version that closely resembles the Consensus-Halving problem. We state the theorem informally.

Theorem 1.6. Necklace-Splitting *and* $\varepsilon$-Consensus-Halving *are computationally equivalent, when* $\varepsilon$ *is inversely polynomial.*

Obviously, Theorem 1.3 then follows from Theorems 1.5 and 1.6. We highlight some interesting recent results about the complexity of the Consensus-Halving problem, which appeared after the conference versions of the papers associated with this work. Filos-Ratsikas et al. [37] showed that the problem is PPAD-hard, even when $\varepsilon$ is constant and one is allowed to use $n+\ell$ cuts, for some fixed $\ell$ that does not depend on the number of agents. The authors of [37] also showed that deciding whether a solution with $n-1$ cuts exists is NP-hard. Filos-Ratsikas et al. [39] showed that the PPA-hardness of the problem is maintained even if we restrict the valuations of the agents to be piecewise uniform functions with only two valuation blocks. Deligkas, Filos-Ratsikas, and Hollender [29] studied the complexity of the problem with a constant number of agents and proved PPA-completeness results and query complexity bounds when the valuation functions are general or monotone, but not necessarily additive measures. Deligkas et al. [28] studied the *exact* version of the problem (when $\varepsilon = 0$) and showed that it is FIXP-hard, whereas deciding whether such a solution with fewer than $n$ cuts exists is ETR-complete. The authors also show that the exact version lies in the newly introduced computational class BU, which captures the complexity of the exact Borsuk–Ulam problem and conjecture that it is actually BU-complete. Batziou, Hansen, and Høgh [8] very recently proved that finding a *strong approximation* of the problem is complete for the appropriate counterpart of BU, the class $\mathrm{BU}_\alpha$.

**1.2.3.** Discrete Ham Sandwich**.** The *Ham Sandwich theorem* [77] is of enduring and widespread interest due to its colorful and intuitive statement and its relevance and applications in topology, social choice theory, and computational

geometry. Roughly, it states that given $d$ measures in Euclidean $d$-space, there exists a hyperplane that cuts them all simultaneously in half. Early work on variants and applications of the theorem focused on nonconstructive existence proofs and mostly did not touch on the algorithmics. A 1983 paper by Hill [50] hints at possible interest in the corresponding computational challenge, in the context of a related land division problem. The computational problem (and its complexity) was first properly studied in a line of work in computational geometry beginning in the 1980s, for example [33, 59, 60, 61]. The problem envisages input data consisting of $d$ sets of $n$ points in Euclidean $d$-space and asks for a hyperplane that splits all point sets in half.

The problem Discrete Ham Sandwich as named in [71] is essentially this, with $d$ set equal to $n$ to emphasize that we care about the high-dimensional case; we provide the definition below.

Definition 1.7 (Discrete Ham Sandwich). *In the* Discrete Ham Sandwich *problem, there are $n$ sets of points in $n$ dimensions having integer coordinates (equivalently one could use rationals). A solution consists of a hyperplane that splits each set of points into subsets of equal size (if any points lie on the plane, we are allowed to place them on either side, or even split them arbitrarily).*

In Definition 1.7, each point set represents an ingredient of the sandwich, which is to be cut by a hyperplane in such a way that all ingredients are equally split.

In the work in computational geometry, the emphasis has been on efficient algorithms for small values of $d$; Lo, Matoušek, and Steiger [60] improve the dependence on $d$ but it is still exponential, and the present paper shows for the first time that we should *not* expect to improve on that exponential dependence. More recently, Grandoni et al. [48] apply the Generalized Ham Sandwich Theorem to a problem in multiobjective optimisation and note that a constructive proof would allow a more efficient algorithm to emerge. The only computational hardness result we know of is [58], which obtains a $W[1]$-hardness result for a constrained version of the problem; [58] points out the importance of the computational complexity of the general problem. The PPA-completeness result of the present paper is the first hardness result *of any kind* for Discrete Ham Sandwich, and as we noted, is a strong notion of computational hardness. Karthik and Saha [56], showing a form of equivalence between the Ham Sandwich Theorem and Borsuk–Ulam, explicitly mention the possible PPA-completeness of Discrete Ham Sandwich as an "interesting and challenging open problem."

We prove the PPA-completeness of Discrete Ham Sandwich via a simple reduction from Necklace-Splitting. Ours is not the first paper to develop the close relationship between the two problems: [13] shows a generalization, where multiple agents may share a "sandwich," dividing it into convex pieces. Further papers to explicitly point out their computational complexity as open problems include [31] (mentioning that both problems "show promise to be complete for PPA"), [2], and [10].

Theorem 1.8. Discrete Ham Sandwich *is* PPA*-complete.*

The proof of the theorem is included in section 7. A limitation to Theorem 1.8 is that the coordinates may be exponentially large numbers; they could not be written in unary. We leave it as an open problem whether a unary-coordinate version is also PPA-complete. As defined in [71], Discrete Ham Sandwich stipulated that each of the $n$ sets of points is of size $2n$, whereas Definition 1.7 allows polynomial-sized sets. We can straightforwardly extend PPA-completeness to the version of [71] by adding "dummy dimensions" whose purpose is to allow larger sets of each ingredient; the new ingredients that are introduced consist of compact clusters of point masses, each

cluster in general position relative to the other clusters and the subspace of dimension $n$ that contains the points of interest.

**1.3. Further related work.** We highlight some rather recent results not directly related to the classes PPA or PPAD, but rather with other subclasses of TFNP. Sotiraki, Zampetakis, and Zirdelis [76] identified the first natural problem for the class PPP, the class of problems whose totality is established by an argument based on the pigeonhole principle. For the class CLS, both Daskalakis, Tzamos, and Zampetakis [25] and Fearnley et al. [36] identified complete problems (two versions of the Contraction Map problem, where a metric or a meta-metric is given as part of the input). In the latter paper, the authors define a new class, namely, EOPL (for "End of Potential Line"), and show that it is a subclass of CLS. Furthermore, they show that two well-known problems in CLS, the P-Matrix Linear Complementarity Problem (P-LCP) and finding a fixpoint of a piecewise-linear contraction map (PL-Contraction), belong to the class. The End of Potential Line problem of [36] is polynomially equivalent to the End of Metered Line of [53]. In a recent breakthrough paper, Fearnley et al. [35] showed that in fact CLS = PPAD $\cap$ PLS and that this class captures the precise complexity of problems related to Gradient Descent. Their result was already used by Babichenko and Rubinstein [7] to prove that computing a *mixed* Nash equilibrium of a congestion game is PPAD $\cap$ PLS-complete.

**2. Overview of the proof.** As we mentioned in the introduction, most of the work in this paper is toward proving Theorem 1.5, as the PPA-completeness of Necklace-Splitting then follows from the equivalence results in section 8. In turn, the PPA-hardness of Necklace-Splitting allows us to establish the PPA-completeness of Discrete Ham Sandwich as well (section 7). In this section, we will provide an overview of the proof of Theorem 1.5, but first, we will present some useful notation and terminology.

*Notation.* We use the standard notation $[n]$ to denote the set $\{1, \ldots, n\}$, and we also use $\pm[n]$ to denote $\{1, -1, 2, -2, \ldots, n, -n\}$. We often refer to elements of $\pm[n]$ as "labels" or "colors." $\lambda$ is usually used to denote a labelling function (so its codomain is $\pm[n]$).

We let $A$ denote the domain of an instance of Consensus-Halving; if that instance has complexity $n$, then $A$ will be the interval $[0, poly(n)]$, where $poly(n)$ is some number bounded by a polynomial in $n$. Recall by Definition 1.4 that $\mu_a$ denotes the value function, or measure, of agent $a$ on the domain $A$, in a Consensus-Halving instance. We also associate each agent with its own cut (recall that the number of agents and cuts is supposed to be equal), and we let $c(a)$ denote the cut associated with agent $a$. We let $p^C(n)$ be a polynomial that represents the number of "circuit-encoders" that we use in our reduction (see subsection 5.1); we usually denote it $p^C$, dropping the $n$ from the notation.

Finally, $B$ denotes the $n$-cube (or "box") $[-1, 1]^n$.

*Terminology.* In an instance of Consensus-Halving, a *value-block* of an agent $a$ denotes a subinterval of the domain where $a$ possesses positive value, uniformly distributed on that interval. In our construction, value-blocks tend to be scattered rather sparsely along the domain.

**2.1. An overview of a simpler reduction.** We start with an overview of a construction which does not quite give us the result that we need, but contains most of the crucial elements of the reduction and is easier to describe.[3]

[3]This reduction was used in our 2018 paper [38], parts of which are contained in this version.

Before presenting this simpler construction, we begin by explaining the ground covered by Filos-Ratsikas et al. [37] (where PPAD-hardness was established), which highlights the challenges when moving from PPAD-hardness to PPA-hardness. In [37], each agent $a$ in a Consensus-Halving instance has a particular cut $c(a)$ associated with $a$. In an instance $I_{CH}$ of Consensus-Halving, we refer to the interval $A$ on which agents have valuation functions, as the *domain* of $I_{CH}$. Filos-Ratsikas et al. [37] established PPAD-hardness by reduction from the PPAD-complete problem $\varepsilon$-Gcircuit ($\varepsilon$-approximate Generalized Circuit), in which the challenge is to find a fixpoint of a circuit in which each node computes (with error at most $\varepsilon$) a real value in the range $[0, 1]$, consisting of a function of at most two other nodes in the circuit; these may be certain simple arithmetic operations or boolean operations (regarding 0 and 1 as representing FALSE and TRUE, respectively). In [37]'s reduction from $\varepsilon$-Gcircuit to Consensus-Halving, each node $\nu$ of a generalized circuit has a corresponding agent $a_\nu$, and the value computed at $\nu$ is represented by the position taken by the cut $c(a_\nu)$. $a_\nu$'s valuation function is designed to enforce the relationship that $\nu$'s value has with the node(s) providing input to $\nu$. Here we reuse some of the circuit "gate gadgets" of [37], in particular the boolean ones. A cut that encodes the value computed at a boolean gate is expected to lie in one of two short intervals, associated with TRUE and FALSE.

To prove PPA-hardness, we will reduce from the computational problem 2D-Tucker, which was recently shown to be PPA-complete by Aisenberg, Bonet, and Buss [2]. In moving from PPAD-hardness to PPA-hardness, however, we encounter a fundamental limitation to the above approach, which is that distinct cuts are constrained to lie in distinct (nonoverlapping) regions of $A$, and *collectively, the cuts lie in an oriented domain* —recall our observations toward the end of section 1.1. A new idea is needed to deal with this issue.

To this end, we construct two special agents (the "coordinate-restricting agents") along with two cuts that correspond to those agents, which are less constrained regarding where, in principle, they may occur, in a solution to the resulting Consensus-Halving instance $I_{CH}$. These two cuts represent the coordinates of a point on a triangular region having two sides identified to form a Möbius strip. In terms of the instance of Consensus-Halving, these cuts are the only ones that lie in a specific subinterval of the interval $A$, called the "coordinate-encoding (c-e) region," and they are called the "coordinate-encoding cuts." Identifying two sides in this way is done by exploiting the equivalence of taking a cut on the LHS of the c-e region and moving it to the RHS.

The rest of $A$ is called the "circuit-encoding region" $R$, and the cuts occurring within $R$ do the job of performing computation on the location of cuts in the c-e region. The reduction uses a sequence of "sensor agents" to identify the endpoints of intervals labelled $A_+$ and $A_-$ in the coordinate-encoding region and feed this information into a set of agents called circuit-encoders, which perform computation on those values. The result of this computation is a label in $\pm[2]$, which is appropriately translated to an assignment of value blocks of the coordinate-restricting agents to either $A_+$ or $A_-$. The assignment is such that the coordinate-restricting agents are satisfied with the discrepancy between the two values if the set of cuts in the coordinate-encoding region corresponds to a point that can be traced back to a solution to 2D-Tucker.

Importantly, the freedom regarding where the coordinate-enconding cuts can occur introduces some new problems, mainly the possibility that one of them may occur outside of the intended "coordinate-encoding region" of the domain of $I_{CH}$; in such a

case, we refer to the cut as a "stray cut." Stray cuts may interfere with the circuitry that $I_{CH}$ uses to encode an instance of 2D-Tucker. We deal with this possibility by making multiple copies of the circuit, so that an unreliable copy is "out-voted" by the reliable ones. The duplication (we use 100 copies) of the circuit serves a further purpose reminiscent of the "averaging maneuver" introduced by [24]; we need to deal with the possibility of values occurring at nodes of the circuit that fail to correspond to boolean values. The duplication corresponds to a sampling of a cluster of points on the Möbius strip, most of which get converted to boolean values.

Another potential problem caused by the stray cuts is that they may change the parity of the labels (the sequence of $A_+$ and $A_-$, potentially affecting the intended behavior of the gadgets corresponding to the circuitry. Fortunately, a "double-negative lemma" guarantees that such a cut is not too harmful, as the effects of this change in parity "cancel out." Finally, we establish that when a cut is moved from one end to the other end of the c-e region, this corresponds to identifying two facets of a simplex to form a Möbius strip.

**2.2. An overview of the actual reduction.** Crucially, the approach above embeds a hard search problem into the surface of a standard two-dimensional Möbius strip, and hence it is necessary to work at exponentially fine resolution. This immediately requires inverse-exponential $\varepsilon$ for instances of $\varepsilon$-Consensus-Halving. While this is enough for establishing the PPA-completeness of $\varepsilon$-Consensus-Halving in general, it falls short of our goal of showing PPA-completness for Necklace-Splitting and Discrete Ham Sandwich, as Theorem 1.6 does not apply unless $\varepsilon$ is inversely polynomial.

To avoid this issue, as the starting-point of the reduction, in section 3 we apply the *snake-embedding* technique invented in [19] (versions of which are used in [30, 31] in the context of PPA) to convert the instance of 2D-Tucker to a grid of fixed resolution, at the expense of going from 2 to $n$ dimensions. The new problem, Variant High-D Tucker (Definition 3.3), envisages a $7 \times 7 \times \cdots \times 7$ grid. Here, we design the snake-embedding in such a way that PPA-completeness holds for instances of the high-dimensional problem that obey a further constraint on the way the high-dimensional grid is colored, which we exploit subsequently. A further variant, New Variant High-D Tucker (Definition 3.5), switches to a "dual" version where a hypercube is divided into cubelets and points in the hypercube are colored such that interiors of cubelets are monochromatic. A pair of points is sought having equal and opposite colors and distant by much less than the size of the cubelets.

We then move to the construction of the Consensus-Halving instance. We encode a point in $n$ dimensions using a solution to an instance of Consensus-Halving as follows. Instead of having just two cuts in the coordinate-encoding region (as in the simpler construction above), suppose we ensure that up to $n$ cuts lie there. These cuts split this interval into $n + 1$ pieces whose total length is constant, so represent a point in the unit $n$-simplex (in the simpler construction, this was the unit 2-simplex). This "Möbius-simplex" (Definition 4.12; Figure 16) has the further property that two facets are identified with each other in a way that effectively turns the simplex into an $n$-dimensional Möbius strip. Note that this is the analogue of the "Möbius triangular region" described in the previous subsection, when generalized to $n$ dimensions. For the aforementioned cuts in the coordinate-encoding region, we now construct $n$ coordinate-restricting agents.

In subsection 5.2 we define a crucially important coordinate transformation (see Figure 17) with the following key properties

- the transformation and its inverse can be computed efficiently, and distances between transformed coordinate vectors are polynomially related to distances between untransformed vectors;
- at the two facets that are identified with each other, the coordinates of corresponding points are the negations of each other; our coloring function (that respects Tucker-style boundary conditions) has the effect that antipodal points get equal and opposite colours, and *no undesired solutions are introduced at these facets.*

This transformation represents a "smooth embedding" of the Möbius-simplex into Consensus-Halving instances: points in the Möbius-simplex map to sequences of cuts in the instance, and "smoothness" refers to the polynomial relationship between corresponding distances.

With the aid of the above coordinate transformation, we divide up the Möbius-simplex:

- The *twisted tunnel* (Definition 5.7) is an inverse-polynomially thick strip, connecting the two facets that are identified in forming the Möbius-simplex. It contains at its center an embedded copy of the hypercube domain of an instance $I_{VT}$ of New Variant High-D Tucker. Outside of this embedded copy, it is "colored in" (using our new coordinate system) in a way that avoids introducing solutions that do not encode solutions of $I_{VT}$.
- The *Significant Region* contains the twisted tunnel and constitutes a somewhat thicker strip connecting the two facets. It serves as a buffer zone between the twisted tunnel and the rest of the Möbius-simplex. It is subdivided into subregions where each subregion has a unique set of labels, or colors, from $\pm[n]$. (We sometimes refer to these as "color-regions".) It is shown that any solution to an instance of Consensus-Halving constructed as in our reduction represents a point in the Significant Region.
- If, alternatively, a set of cuts represents a point from outside the Significant Region, then certain agents (so-called tunnel-boundary sensor agents) will observe severe imbalances between labels $A_+$ and $A_-$, precluding a solution. This is achieved via appropriate "feedback" that is provided to the coordinate-restricting agents, which results in them being dissatisfied with the balance of $A_+$ and $A_-$.

When working in two dimensions, it is relatively straightforward to integrate the subset of the two-dimensional Möbius-simplex that corresponds with the twisted tunnel, with the parts of the domain where the tunnel-boundary sensoragents become active (ruling out a solution) in a way that avoids introducing solutions that fail to encode solutions of Tucker. In fact, the construction of the previous subsection requires only a single tunnel-boundary sensoragent. In $n$ dimensions, that gap has to be colored-in in a carefully designed way (subsection 5.3, list item 3), and this is the role of the part of the Significant Region that is not the twisted tunnel. The proofs that they work correctly (subsections 6.2 and 6.3) become more complicated.

As in the simpler construction, we use multiple copies of the circuit that performs the computation, each in its own subregion of $R$. Here we use $p^C(n)$ copies where $p^C$ is a polynomial, instead of 100 copies. Each copy is called a *circuit-encoder*. As before, the purpose of multiple copies is to make the system robust; a small fraction of copies may be unreliable, and we have to account for the possibility that one of the c-e cuts may occur in the circuit-encoding region, rendering one of the copies unreliable. The "double-negative lemma" of the simpler construction applies here as well, and again we can ensure that when a cut is moved from one end to the other end of the

c-e region, this corresponds to identifying two facets of the simplex to form a Möbius strip.

**3. Snake embedding reduction.** The purpose of this section is to establish the PPA-completeness of New Variant High-D Tucker, Definition 3.5. The snake embedding construction was devised in [19], in order to prove that $\varepsilon$-Nash equilibria are PPAD-complete to find when $\varepsilon$ is inverse polynomial; without this trick the result is just obtained for $\varepsilon$ being inverse exponential. We do a similar trick here. We will use as a starting-point the PPA-completeness of 2D-Tucker, from [2], which is the following problem.

Definition 3.1 (Aisenberg et al. [2]). *An instance of* $2D$-Tucker *consists of a labelling* $\lambda : [m] \times [m] \to \{\pm 1, \pm 2\}$ *(which is given implicitly as a labelling boolean circuit) 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 of* $2D$-Tucker *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 hardness of the problem in Definition 3.1 arises when $m$ is exponentially-large, and the labelling function is presented by means of a boolean circuit.

We aim to prove the following is PPA-complete, even when the values $m_i$ are all upper-bounded by some constant (specifically, 7).

Definition 3.2 (Aisenberg, Bonet, and Buss [2]). *An instance of* $nD$-Tucker *consists of a labelling* $\lambda : [m_1] \times \cdots \times [m_n] \to \{\pm 1, \ldots, \pm n\}$ *(which is given implicitly as a labelling boolean circuit) such that if a point* $\mathbf{x} = (x_1, \ldots, x_n)$ *lies on the boundary of this grid (i.e.,* $x_i = 1$ *or* $x_i = m_i$ *for some* $i$*), then letting* $\bar{\mathbf{x}}$ *be the antipodal point of* $\mathbf{x}$*, we have* $\lambda(\bar{\mathbf{x}}) = -\lambda(\mathbf{x})$. *(Two boundary points are antipodal if they lie at opposite ends of a line segment passing through the center of the grid.) A solution consists of two points* $\mathbf{z}$, $\mathbf{z}'$ *on this grid, having opposite labels* ($\lambda(\mathbf{z}) = -\lambda(\mathbf{z}')$)*, each of whose coordinates differ (coordinatewise) by at most* 1.

*It is assumed that* $\lambda$ *is presented in the form of a circuit, syntactically constrained to give opposite labels to antipodal grid points.*

Definition 3.3. *An instance of* Variant High-D Tucker *is similar to Definition* 3.2 *but whose instances obey the following additional constraints. The* $m_i$ *are upper bounded by the constant* 7. *We impose the further constraint that the facets of the cube are colored with labels from* $\pm[n]$ *such that all colors are used, and opposite facets have opposite labels, and for* $2 \leq i \leq n$ *it holds that the facet with label* $i$ *(respectively,* $-i$*) has no grid-point on that facet with label* $i$ *(respectively,* $-i$*).*

Theorem 3.4. Variant High-D Tucker *is* PPA*-complete.*

*Informal description of snake embedding.* A snake-embedding consists of a reduction from $k$D-Tucker to $(k+1)$D-Tucker, which we describe informally as follows. See Figure 1. Let $I$ be an instance of $k$D-Tucker, on the grid $[m_1] \times \cdots \times [m_k]$. Embed $I$ in $(k+1)$-dimensional space, so that it lies in the grid $[m_1] \times \cdots \times [m_k] \times [1]$. Then sandwich $I$ between two layers, where all points in the top layer get labelled $k+1$, and points in the bottom layer get labelled $-(k+1)$, as in the left part of Figure 1. We now have points in the grid $[m_1] \times \cdots \times [m_k] \times [3]$, and notice that this construction preserves the required property that points on the boundary have labels opposite to their antipodal points.

Then, the main idea of the snake embedding is the following. We fold this grid into three layers, by analogy with folding a sheet of paper along two parallel lines so that the cross section is a zigzag line, and one dimension of the folded paper is

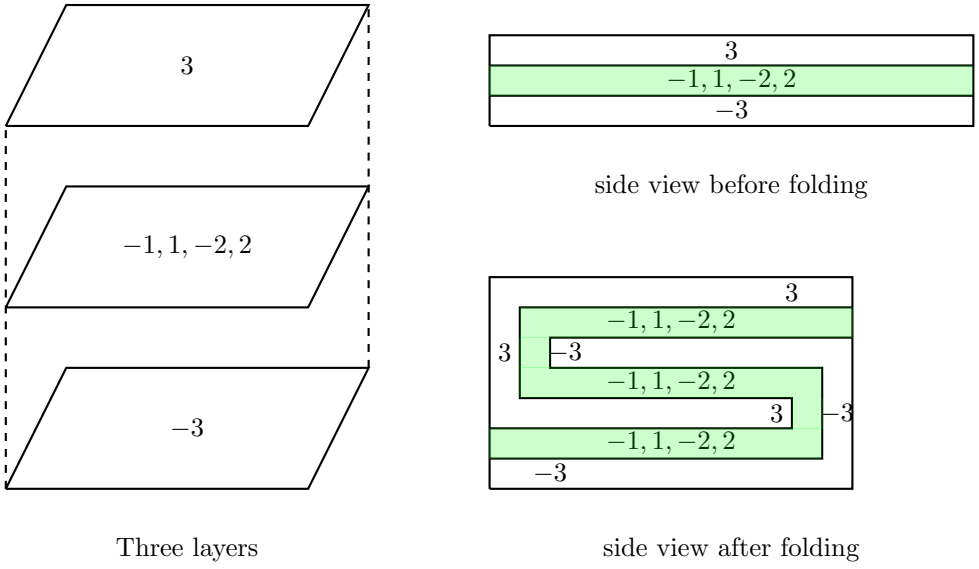


FIG. 1. *Snake embedding from two to three dimensions.*

one-third of the unfolded version, the other dimension being unchanged (see the RHS of Figure 1). In higher dimension, suppose that $m_1$ is the largest value of any $m_i$. Then, we can reduce $m_1$ by a factor of about 3, while causing the final coordinate to go up from 3 to 9. By merging layers of label $k+1$ and $-(k+1)$, the thickness of 9 reduces to 7. *This operation preserves the labelling rule for antipodal boundary points.*

However, there are two points that need extra care for the reduction to go through:

- First, simply folding the layers such that their cross sections are zigzag lines may introduce diagonal adjacencies between cubelets that were not present in the original instance in $k$-dimensions, i.e., we might end up generating adjacent cubelets with equal-and-opposite colours; see the left part of Figure 2 for an illustration. To remedy this, we will "copy" (or "duplicate") the cubelets at the folding points, essentially having three cubelets of the same color, whose cross sections are the short vertical section in the RHS of Figure 1; see also the RHS of Figure 2 for an illustration. From now on, when referring to "folding," we will mean the version where we also duplicate the cubelets at the folding points, as described above.
- Second, the folding and duplicating operation only works if $m_1$ is a multiple of 3, as otherwise the $(k+1)$-dimensional instance may not satisfy the boundary conditions of Definition 3.2, i.e., we might end up with antipodal cubelets that do not have equal-and-opposite colors. To ensure that $m_1$ is a multiple of 3 before folding, we can add one or two additional layers of cubelets to $m_1$ (depending on whether the remainder of the division of $m_1$ by 3 is either 2 or 1, respectively). These layers are duplicate copies of the outer layers of cubelets at opposite ends of the length-$m_1$ direction; if there is only one additional layer to be added, we can add on either side. Note that this operation does not generate any cubelets of equal-and-opposite labels that were not there before and the same will be true for the instance after the folding operation. See Figure 3 for an illustration.

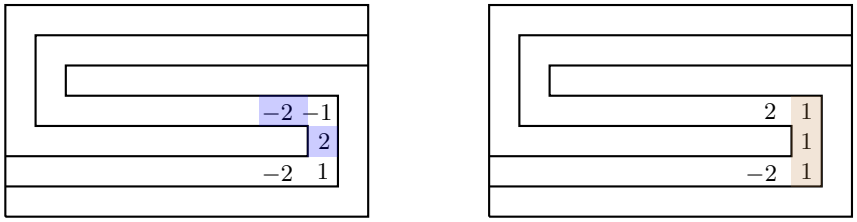


FIG. 2. *Side views of the folding operation with and without duplications of cubelets. On the left, simply folding generates equal-and-opposite labels diagonally in the shaded cubelets. On the right, the duplication of the cubelet at the folding position in three copies prevents this from happening.*

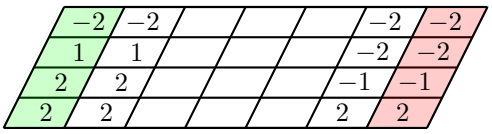


FIG. 3. *Extending the coloring to ensure that $m_1$ is a multiple of 3. In the figure, the case when $m_1 \mod 3 = 1$ is shown, i.e., one layer needs to be added at each side.*

*Formal description of snake embedding.* Let $I$ be an instance of $k$D-TUCKER having coordinates in ranges $[m_1], \ldots, [m_k]$ and label function $\lambda$. Select the largest $m_i$, breaking ties lexicographically. Assume for simplicity in what follows that $m_1$ is largest.

**Fixing the length to a multiple of** 3. Let $r = m_1 \mod 3$ and let $\ell = 3 - r$. Consider the instance $I^3$ of $kD$-TUCKER having coordinates in ranges $[m'_1], \ldots, [m_k]$, with $m'_1 \mod 3 = 0$, constructed from $I$ as follows. For any point $\mathbf{x}' = (x'_1, \ldots, x'_k)$ in $[m'_1] \times \cdots \times [m_k]$, $\mathbf{x}'$ is mapped to a point $\mathbf{x}$ in $[m_1] \times \cdots \times [m_k]$ and receives a colour $\lambda'(\mathbf{x}')$ such that the following hold:

- If $\ell = 0$, then $\mathbf{x}'$ is mapped to $x = (x'_1, \ldots, x'_k)$ and $\lambda'(\mathbf{x}') = \lambda(\mathbf{x})$, i.e., $\mathbf{x}'$ is mapped to itself and receives its own label, since $m_1$ is already a multiple of 3.
- If $\ell = 1$, then
  - If $x'_1 \leq m_1$, $\mathbf{x}'$ is mapped to $\mathbf{x} = (x'_1, \ldots, x'_k)$ and $\lambda'(\mathbf{x}') = \lambda(\mathbf{x})$;
  - If $x'_1 = m_1 + 1$, $\mathbf{x}'$ is mapped to $\mathbf{x} = (m_1, \ldots, x'_k)$ and $\lambda'(\mathbf{x}') = \lambda(\mathbf{x})$.

  In other words, points for which the first coordinate ranges from 1 to $x'_1$ are mapped to themselves and receive their own label, and points for which the first coordinate is $m_1 + 1$ are mapped to the points where the first coordinate is $m_1$, receiving the label of that point. This essentially "duplicates" the layer of cubelets on the right endpoint of the $m_1$-direction. See Figure 3 for an illustration.
- If $\ell = 2$, then
  - if $x'_1 = 1$, $\mathbf{x}'$ is mapped to $\mathbf{x} = (1, \ldots, x'_k)$ and $\lambda'(\mathbf{x}') = \lambda(\mathbf{x})$;
  - if $2 \leq x'_1 \leq m_1 + 1$, $\mathbf{x}'$ is mapped to $\mathbf{x} = (x'_1 - 1, \ldots, x'_k)$ and $\lambda'(\mathbf{x}') = \lambda(\mathbf{x})$. This is similar to the mapping and labelling in the previous case, except for the fact that we need to "shift" the labels of the points, since we essentially introduced a copy of the layer of cubelets on the left endpoint of the $m_1$-direction. See Figure 3 for an illustration.

Note that by the operation of adding $\ell$ layers as above, we do not introduce any cubelets with equal-and-opposite labels that were not present before. To avoid complicating the notation, in the following we will use $m_1$ to denote the maximum size of the first coordinate (instead of $m_1'$) and we will assume that $m_1$ is a multiple of 3. We will also use $I$ to denote the instance of $kD$-TUCKER where $m_1$ is a multiple of 3, instead of $I^3$ as denoted above.

**From $k$ to $k+1$ dimensions**. Starting from an instance $I$ of $kD$-TUCKER, we will construct an instance $I'$ of $(k+1)D-$TUCKER as follows. Let $\mathbf{x} = (x_1, \ldots, x_k)$ be a point in $[m_1] \times \cdots \times [m_k]$ with labelling function $\lambda$. We will associate each such point with a corresponding point $\mathbf{x}'$ in $\left[\frac{m_1}{3} + 2\right] \times \cdots \times [m_k] \times [7]$ and a label $\lambda'(\mathbf{x}')$ as follows:

- If $x_1 \leq \frac{m_1}{3}$, then $\mathbf{x}$ is mapped to $\mathbf{x}' = (x_1, \ldots, x_k, 2)$, and $\lambda'(\mathbf{x}') = \lambda(\mathbf{x})$.
- If $x_1 = \frac{m_1}{3} + 1$ *(the first "folding" point)*, then $\mathbf{x}$ is mapped to the following three points in $I'$ and receives the following colors (see the shaded cubelets at the RHS of Figure 2):
  - $\mathbf{x}' = (\frac{m_1}{3} + 1, \ldots, x_k, 2)$ *(the original cubelet)* and $\lambda'(\mathbf{x}') = \lambda(\mathbf{x})$;
  - $\mathbf{x}' = (\frac{m_1}{3} + 1, \ldots, x_k, 3)$ *(the first copy)* and $\lambda'(\mathbf{x}') = \lambda(\mathbf{x})$;
  - $\mathbf{x}' = (\frac{m_1}{3} + 1, \ldots, x_k, 4)$ *(the second copy)* and $\lambda'(\mathbf{x}') = \lambda(\mathbf{x})$.
- If $\frac{m_1}{3}+2 \leq x_1 \leq \frac{2m_1}{3}-1$, then $\mathbf{x}$ is mapped to $\mathbf{x}' = (\frac{2m_1}{3}+2-x_1, x_2, \ldots, x_k, 4)$ with $\lambda'(\mathbf{x}') = \lambda(\mathbf{x})$.
- If $x_1 = \frac{2m_1}{3}$ *(the second folding point)*, then $\mathbf{x}$ is mapped to the following three points in $I'$ and receives the following colors:
  - $\mathbf{x}' = (2, \ldots, x_k, 4)$ *(the original cubelet)* and $\lambda'(\mathbf{x}') = \lambda(\mathbf{x})$;
  - $\mathbf{x}' = (2, \ldots, x_k, 5)$ *(the first copy)* and $\lambda'(\mathbf{x}') = \lambda(\mathbf{x})$;
  - $\mathbf{x}' = (2, \ldots, x_k, 6)$ *(the second copy)* and $\lambda'(\mathbf{x}') = \lambda(\mathbf{x})$.
- If $\frac{2m_1}{3} + 1 \leq x_1 \leq m_1$, then $\mathbf{x}$ is mapped to $\mathbf{x}' = (x_1 + 2 - \frac{2m_1}{3}, x_2, \ldots, x_k, 6)$ with $\lambda'(\mathbf{x}') = \lambda(\mathbf{x})$.
- Set $\lambda'(1, \ldots, 1) = -k$, along with any point $\mathbf{x}$ connected to it via a path of points that have not been labelled by the above procedure.
- Set $\lambda'(\frac{m_1}{3} + 2, m_2, \ldots, m_k, 7) = k$, along with any point $\mathbf{x}$ connected to it via a path of points that have not been labelled by the above procedure.

We are now ready to prove Theorem 3.4.

*Proof of Theorem* 3.4. First, it is not hard to check that the composition of $O(n)$ snake-embeddings is a polynomial-time reduction, where by "composition" here we refer to the implicit composition based on the labelling circuits. Also note that, by the way the high-dimensional instance is constructed, we have not introduced any adjacencies that did not already exist, i.e., if there is a pair of adjacent cubelets with equal-and-opposite labels in the instance $I'$ of the high-dimensional version, this pair is present in the instance $I$ of the two-dimensional version as well, and it is easy to recover it in polynomial time. Therefore, it suffices to show how to obey the additional constraint of VARIANT HIGH-D TUCKER, namely that for $i \geq 2$, a side having label $i$ has no grid-points with label $i$, and similarly for $-i$.

We begin by taking the original two-dimensional instance $I$, of size $m \times m$, and extend to an instance of size $3m \times 3m$ as follows. The original instance is embedded in the center of the new instance. Each region $R$ to the sides (of size $m \times m$) are labelled by copying the edge of $I$ facing $R$, along an adjacent edge of $R$, and connecting these two edges with paths that have two straight sections and connect two points of the same label, and points along that path have that label. The outermost path then labels a side of the new instance of length $m$, so these two opposite sides get opposite

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | -2 | 1 | 2 | 2 | 1 | -2 | -2 | -2 | -2 | -2 | -2 |
| 2 | -2 | 1 | 2 | 2 | 1 | -2 | -2 | -2 | -2 | -2 | -2 |
| 2 | -2 | 1 | 2 | 2 | 1 | -2 | -2 | -2 | -2 | -2 | -2 |
| 2 | -2 | 1 | 2 | 2 | 1 | -2 | -2 | -2 | -2 | -2 | -2 |
| 2 | -2 | 1 | 2 | 2 | 1 | -2 | -2 | -2 | -2 | -2 | -2 |
| 2 | -2 | 1 | 1 | 1 | -2 | -2 | -2 | -2 | -2 | -2 | -2 |
| 2 | -2 | -2 | -2 | -2 | -2 | -1 | -1 | -1 | -1 | -2 | -2 |
| 2 | 2 | 2 | 2 | 2 | 2 | -1 | -2 | -2 | -1 | -2 | -2 |
| 2 | 2 | 2 | 2 | 2 | 2 | -1 | -2 | -2 | -1 | -2 | -2 |
| 2 | 2 | 2 | 2 | 2 | 2 | -1 | -2 | -2 | -1 | -2 | -2 |
| 2 | 2 | 2 | 2 | 2 | 2 | -1 | -2 | -2 | -1 | -2 | -2 |
| 2 | 2 | 2 | 2 | 2 | 2 | -1 | -2 | -2 | -1 | -2 | -2 |

FIG. 4. *The extension from an $m \times m$ instance of 2D-*TUCKER *to a $3m \times 3m$ instance, where opposite sides have equal-and-opposite labels.*

labels. We may assume (by switching 1's and 2's if needed) that these new opposite sides are labelled $\pm 2$. See Figure 4.

The $S$-fold approach shown in Figure 1 can be checked to retain this property. When we sandwich a cuboid between two layers of opposite (new) colors (call them $c$ and $-c$), as shown in Figure 1, we label the new facets thus formed with $-c$ and $c$, respectively. We label the other facets with their original labels (each of these facets has acquired the labels $c$ and $-c$, and no other labels). The folding operation has a natural correspondence between the facets of the unfolded and folded versions of the cuboid. It can be checked that the set of colors of a facet before folding is the same as the set of colors of the corresponding facets after folding. ∎

It is convenient to define the following problem, whose PPA-completeness follows fairly directly from the PPA-completeness of VARIANT HIGH-D TUCKER.

DEFINITION 3.5. *An instance of* NEW VARIANT HIGH-D TUCKER *in $n$ dimensions is presented by a boolean circuit $C_{VT}$ whose input consists of coordinates of a point in the hypercube $B = [-1, 1]^n$ and whoose output is a label in $\pm[n]$ (assume $C_{VT}$ has $2n$ output gates, one for each label, and is syntactically constrained such that exactly one output gate will evaluate to* TRUE*), having the following constraints that may be enforced syntactically:*

1. *Dividing $B$ into $7^n$ cubelets of edge length $2/7$ using axis-aligned hyperplanes, all points in the same cubelet get the same label by $C_{VT}$.*
2. *Interiors of antipodal boundary cubelets get opposite labels.*
3. *Points on the boundary of two or more cubelets get a label belonging to one of the adjacent cubelets.*
4. *Facets of $B$ are colored with labels from $\pm[n]$ such that all colors are used, and opposite facets have opposite labels. For $2 \leq i \leq n$ it also holds that the facet with label $i$ (respectively, $-i$) does not intersect any cubelet having label $i$ (respectively, $-i$). Facets colored $\pm 1$ are unrestricted (we call them the "panchromatic facets").*

*A solution consists of a pair of points that lie within an inverse polynomial distance $\delta(n)$ of each other (for concreteness, assume $\delta(n) = \frac{1}{100n}$). Those points should receive equal and opposite labels by $C_{VT}$.*

**4. Some building-blocks and definitions.** Here we set up some of the general structure of instances of Consensus-Halving constructed in our reduction. We identify some basic properties of solutions to these instances. We define the Möbius-simplex and the manner in which a solution encodes a point on the Möbius-simplex. The detailed encoding of the circuitry is covered in section 5.

*Useful quantities.* We use the following values throughout the paper.

- $\delta^{\text{tiny}}$ is an inverse-polynomial quantity in $n$, chosen to be substantially smaller than any other inverse-polynomial quantity that we use in the reduction, apart from $\varepsilon$ (below).
- $\delta^T$ is an inverse-polynomial quantity in $n$, which is smaller than any other inverse-polynomial quantity apart from $\delta^{\text{tiny}}$ and is larger than $\delta^{\text{tiny}}$ by an inverse-polynomial amount. The quantity $\delta^T$ denotes the width of the so-called "twisted tunnel" (see Definition 5.7).
- $p^{\text{huge}}$ denotes a large polynomial in $n$; specifically, we let $p^{\text{huge}} = n/\delta^{\text{tiny}}$. The quantity $p^{\text{huge}}$ represents the number of sensor agents for each circuit encoder (see Definition 4.5).
- $p^{\text{large}}$ denotes a large polynomial in $n$, which is, however, smaller than $p^{\text{huge}}$ by a polynomial factor. The quantity $p^{\text{large}}$ will be used in the definition of the tunnel-boundary sensor agents (see Definition 4.6) and will quantify the extent to which the cuts in the coordinate-encoding region (Definition 4.1) are allowed to differ from being evenly spaced, before the tunnel-boundary sensor agents become active (see section 4). The choice of $p^{\text{large}}$ controls the value $\delta_w$ of the radius of the Significant Region (see Proposition 4.15), with larger $p^{\text{large}}$ meaning larger $\delta_w$.
- $\varepsilon$ is the precision parameter in the Consensus-Halving solution, i.e., each agent $i$ is satisfied with a partition as long as $|\mu_i(A_+) - \mu_i(A_-)| \leq \varepsilon$. Henceforth, we will set $\varepsilon = \delta^{\text{tiny}}/10$.

  Given the above, we have the following qualitative relations between these quantities:

$$\varepsilon \ll \delta^{\text{tiny}} \ll \delta^T \ll p^{\text{large}} \ll p^{\text{huge}}.$$

**4.1. Basic building-blocks.** We consider instances $I_{CH}$ of the Consensus-Halving problem that have been derived from instances $I_{VT}$ of New Variant High-D Tucker in $n$ dimensions. The general aim is to get any solution of such an instance $I_{CH}$ to encode a point in $n$ dimensions that "localizes" a solution to $I_{VT}$, by which we mean that from the solution of $I_{CH}$, we will be able to find a point on the $I_{VT}$ instance that can be transformed to a solution of $I_{VT}$ in polynomial time and fairly straightforwardly.

We start with the definition of the coordinate-encoding region and the corresponding agents.

Definition 4.1. Coordinate-encoding region (c-e region). *Given some instance of* Variant High-D Tucker *in $n$ dimensions, the corresponding instance of* Consensus-Halving *has a* coordinate-encoding region*, the interval $[0, n]$, a (prefix) subinterval of $A$ (see Figure 5).*

The valuation functions of agents in an instance $I_{CH}$ of Consensus-Halving obtained by our reduction from an instance of New Variant High-D Tucker in $n$ dimensions will be designed in such a way that either $n-1$ or $n$ cuts (typically $n$) must occur in the coordinate-encoding region, in any solution. Furthermore, the

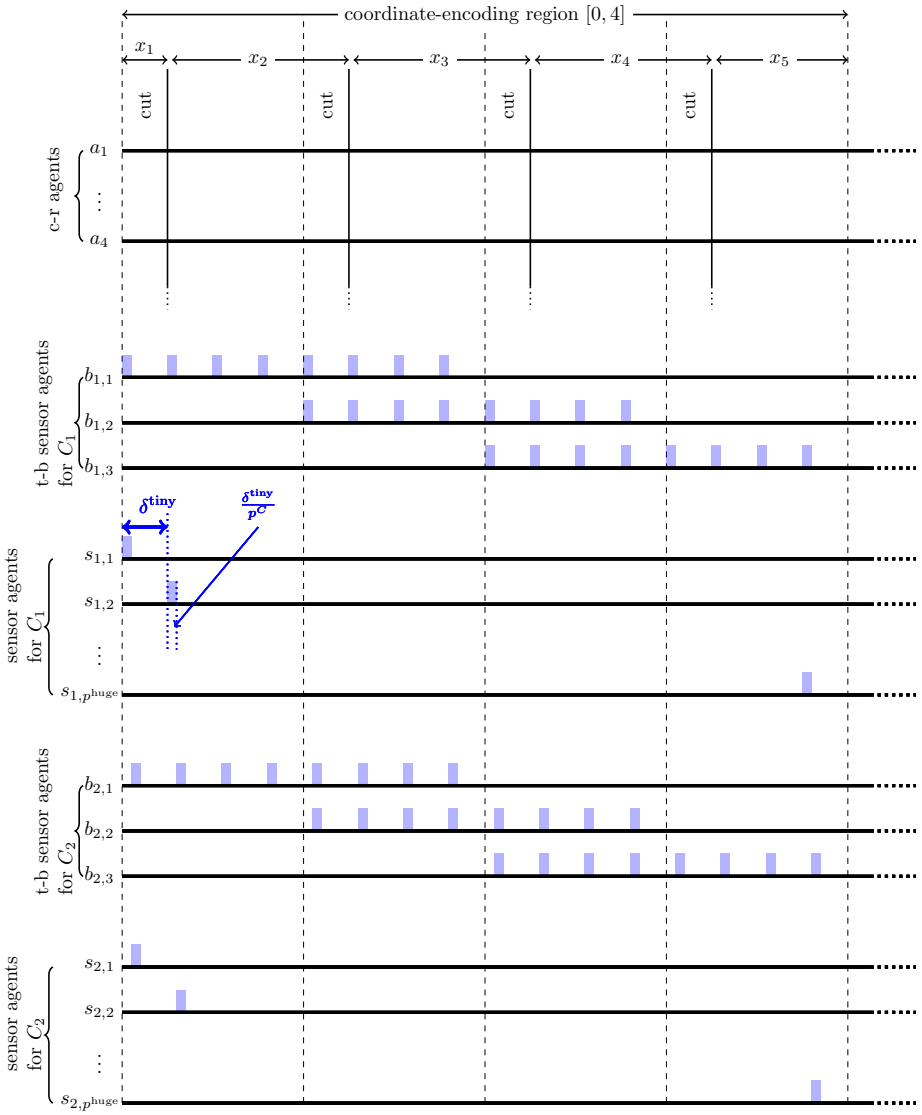


FIG. 5. *Sensor illustration: example of $n = 4$ c-e cuts representing five coordinates summing to* 1 *(a typical point in the Möbius-simplex). Vertical lines depict the cuts, resulting in labels that alternate between $A_+$ and $A_-$, starting with $A_+$. Shaded blocks over agents' lines indicate value-blocks of their value functions. We only depict sensors for circuit-encoders $C_1$ and $C_2$.*

distance between consecutive cuts must be close to 1 (an additive difference from 1 that is upper-bounded by an inverse polynomial), shown in Proposition 4.15.

DEFINITION 4.2. Coordinate-restricting agents (c-r agents). *Given an instance of* NEW VARIANT HIGH-D TUCKER *in $n$ dimensions, the corresponding instance of* CONSENSUS-HALVING *has $n$* coordinate-restricting agents *denoted $\{a_1, \ldots, a_n\}$ (see Figure 5). Each of the c-r agents has—for each of the $p^C$ copies of the circuit—two value-blocks of value $1/(2p^C)$ outside the c-e region and inside the circuit-encoding region, in positions that overlap with the value-blocks of the outputs of the simulated circuit; see subsections* 5.1.2 *and* 5.1.4 *for the details.*

The $n$ coordinate-restricting agents have associated $n$ coordinate-encoding cuts (Remark 4.3). It will be seen that the c-e cuts typically occur in the c-e region. The c-r agents do *not* have any value for the coordinate-encoding region; their value functions are only ever positive elsewhere. In particular, they have blocks of value whose labels $A_+/A_-$ are affected by the output gates of the circuitry that is encoded to the right of the c-e region.

*Remark* 4.3. *Coordinate-encoding cuts (c-e cuts).* We identify $n$ cuts as the coordinate-encoding cuts. In the instances of Consensus-Halving that we construct, in any (sufficiently good approximate) solution to the Consensus-Halving instance, all other cuts will be constrained to lie outside the c-e region (and it will be straightforward to see that the value functions of their associated agents impose this constraint). A c-e cut is not straightforwardly constrained to lie in the c-e region, but it will ultimately be proved that in any approximate solution, the c-e cuts do in fact lie in the c-e region.

Recall that $p^{\text{huge}} = n/\delta^{\text{tiny}}$, which implies that the c-e region can be divided into $p^{\text{huge}}$ intervals of length $\delta^{\text{tiny}}$ (see also Figure 5).

*Circuit-encoders.* Recall that the circuit-encoding region (detailed in section 5) contains $p^C$ circuit-encoders, which we describe in this subsection. Before we do that, we introduce some useful gadgets, the bit-detection gadgets.

*Bit detection gadgets.* The ability to detect the position of the cuts in the c-e region and feed this information to the circuit will lie in the presence of gadgets developed in [37], which we refer to as bit detection gadgets. A bit detection gadget consists typically of two *thin* and *dense* valuation blocks of relatively large height and relatively small length, situated next to each other (e.g., see the rightmost set of value-blocks in Figure 6). These value-blocks constitute most of the agent's valuation over the related interval. The point of these gadgets is that if the discrepancy between $A_+$ and $A_-$ is (significantly) in favor of one against the other, there will be a cut

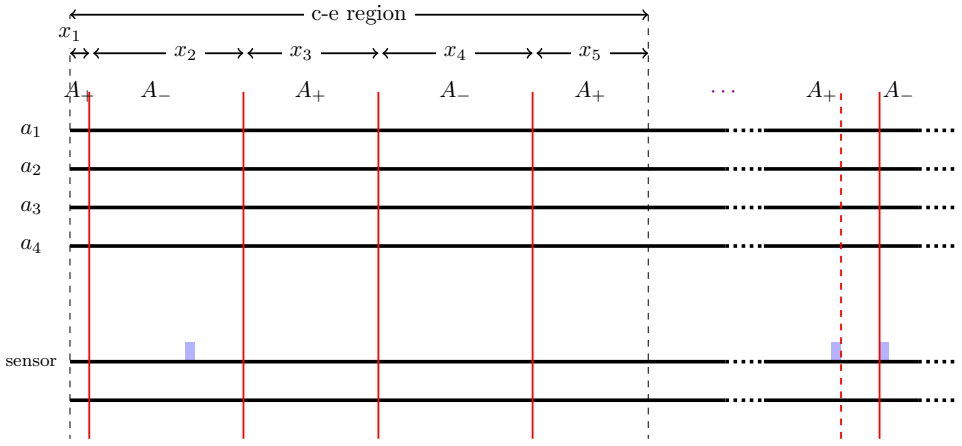


Fig. 6. *An example of how the input of a sensor agent is processed into a boolean value that will be used by the encoding of the circuit. One of the two value-blocks on the RHS of the picture (the bit-detection gadget) is intersected by the cut corresponding to the sensor, depending on whether the value-block on the LHS is labelled $A_+$ or $A_-$. In the figure, the block is labelled $A_-$ and therefore the cut intersects the rightmost value-block on the RHS. The other (unused) option is depicted by a red dashed line.*

intersecting one of the two valuation blocks; which block is intersected will correspond to a 0/1 value, i.e., a bit that indicates the "direction" of the discrepancy in the two labels. These gadgets are used in several parts of the reduction.

Each circuit encoder $C_i$ will consist of the following sets of agents:

- The *sensor agents* $\mathcal{S}_i$ that are responsible for extracting the positions of the cuts in the c-e region, which will be used as inputs to the other circuit-encoding agents. These agents have value in the c-e region, as well as in a region $R_i$ of the circuit-encoding region, which does not overlap with any other region $R_j$ for any other circuit-encoder.
- The *tunnel-boundary sensor agents* (or simply *tunnel-boundary sensors*), responsible for detecting large discrepancies between the balances of the two labels $A_+$ and $A_-$ in the c-e region, preventing a solution in such a case. These agents will also have value in the c-e region and in the region $R_i$.
- The *gate agents* $\mathcal{G}_i$ that implement the labelling circuit $C_{VT}$ of New Variant High-D Tucker.

Definition 4.4. $\sigma$-shifted version. *Given a value function $f$ (or measure) on the c-e region $[0, n]$, we say that another function $f'$ on the c-e region is a* $\sigma$-shifted version *of $f$, when we have that $f'((x-\sigma) \mod n) = f(x)$.*

Now we are ready to define the sensor agents.

Definition 4.5. Sensor agents. *Each circuit-encoder $C_i$, $i = 1, \ldots, p^C$, has a set $\mathcal{S}_i$ of sensor agents, $\mathcal{S}_i = \{s_{i,1}, \ldots, s_{i,p^{\text{huge}}}\}$, where the $s_{i,j}$ are defined as follows. When $i = 1$, $s_{1,j}$ has value $\frac{1}{10}$ uniformly distributed over the interval*

$$\left[(j-1)\delta^{\text{tiny}}, (j-1)\delta^{\text{tiny}} + \frac{\delta^{\text{tiny}}}{p^C}\right].$$

*For $i > 1$, $s_{i,j}$ is a $\frac{1}{p^C}(i-1)\delta^{\text{tiny}}$-shifted version of $s_{1,j}$.*

*Each sensor agent $s_{i,j}$ also has valuation outside the c-e region, in nonoverlapping intervals of the circuit-encoding region $R_i$ (see subsection 5.1). This valuation consists of two valuation blocks of value $\frac{9}{20}$ each, with no other valuation block in between, i.e., a bit-detection gadget.*

*This value gadget for $s_{i,j}$ causes the $j$th input gate in the circuit-encoder $C_i$ to be set according to the label received by $s_i$'s block of value in the c-e region, i.e., jump to the left or to the right in order to indicate that the corresponding value-block of $s_i$ in the c-e region is labelled $A_+$ or $A_-$. See Figure 6 for an illustration.*

According to the definitions above, $C_1$ has a sequence of (a large polynomial number of) sensor agents that have blocks of value in a sequence of small intervals going from left to right of the c-e region (see Figure 5). For $1 < i \leq p^C$, $C_i$ has a similar sequence, shifted slightly to the right on the c-e region (by $\delta^{\text{tiny}}(i-1)/p^C$). For $j \in [p^{\text{huge}}]$, the intervals defined by the value-blocks of the sensor agents $s_{1,j}, \ldots, s_{p^C,j}$ (for $C_1, \ldots, C_{p^C}$) partition the interval $[(j-1)\delta^{\text{tiny}}, j\delta^{\text{tiny}}]$.

*Remark.* Note that a c-e cut may divide one of the above value-blocks held by a sensor agent in the c-e region, and in that case the input being supplied to its circuit-encoder is *unreliable*. However, only $n$ sensor agents may be affected in that way, and their circuit-encoders will get out-voted by the ones that receive valid boolean inputs. This is part of the reason why we use $p^C$ circuit-encoders in total. More details on this averaging argument are provided in section 5.

Notes on the sensor agents. Looking a bit ahead, here we provide some additional intuition on the operation of the sensor agents. Each such agent of $S_i$ has a small

value-block (of value 1/10) in the c-e region and its remaining value (9/10) lies in the circuit encoding region, and particularly, in the subregion $R_i$ (as we explain in subsection 5.1, the circuit-encoding region $R$ is partitioned into subregions $R_i$, one for each circuit encoder $C_i$, where most of the gadgetry of the encoder lies). In particular, the sensor agent has two thin blocks of value 9/20 in the c-e region and this is precisely the bit detection gadget of the agent. If the value-block on the LHS (in the c-e region) is labelled $A_-$, then the cut on the RHS intersects the rightmost value-block (i.e., jumps to the right) and if it is $A_+$, then it jumps to the left. This information is then passed on to the next level of circuit encoding agents, those that implement the circuit. These inputs are then "propagated" through the encoding of the circuit $C_{VT}$, to produce the appropriate labels at the output gates $g_j$, $j \in \pm[n]$, as described in subsection 5.1.2.

Next, we define the tunnel-boundary sensors.

DEFINITION 4.6. Tunnel-boundary sensor agents. *Each circuit-encoder $C_i$ shall have $n-1$* tunnel-boundary sensor agents $b_{i,2}, \ldots, b_{i,n}$.

1. *In $C_1$, for each $j = 2, \ldots, n$, tunnel-boundary sensor agent $b_{1,j}$ has value* 1/10 *distributed over $[j-2, j]$ (see Figure* 5*). This value consists of a sequence*
$$\left[(j-2), (j-2) + \frac{\delta^{\text{tiny}}}{p^C}\right], \ldots, \left[j - \delta^{\text{tiny}}, j - \delta^{\text{tiny}} + \frac{\delta^{\text{tiny}}}{p^C}\right]$$
*of $2/\delta^{\text{tiny}} = 2p^{\text{huge}}/n$ value-blocks, each of length $\delta^{\text{tiny}}/p^C$ and of value $\frac{1}{10} \cdot (\delta^{\text{tiny}}/2)$.*
2. *In each $C_i$, $1 < i \le p^C$, and for each $j = 2, \ldots, n$, the value function of $b_{i,j}$ that lies in the c-e region is an $(i-1)\frac{\delta^{\text{tiny}}}{p^C}$-shifted version of $b_{1,j}$.*
3. *The remaining value* 9/10 *of each $b_{i,j}$ consists of three value-blocks of width $\delta^{\text{tiny}}$ lying in a subinterval $I_{i,j}$ of the circuit-encoding region $R_i$ (see subsection* 5.1*), such that*
   - *the value-blocks have values*
$$\frac{9(1-\kappa)}{20}, \quad \frac{9\kappa}{10}, \quad \frac{9(1-\kappa)}{20},$$
   *respectively, where $\kappa = (\frac{1}{10}\frac{\delta^{\text{tiny}}}{2})p^{\text{large}}$;*
   - *$I_{i,j}$ contains also value-blocks of agents for each gate that takes the value of $b_{i,j}$ as input (the feedback gadgetry; see subsection* 5.1.2*).*

For an illustration, see Figure 7. The structure of the tunnel-boundary sensor agents in the c-e region is shown in Figure 5.

Notes on the tunnel-boundary sensors. Each tunnel-boundary sensor agent $b_{i,j}$ has an associated cut $c(b_{i,j})$ that lies in the subinterval $I_{i,j}$. Agent $b_{i,j}$ "monitors" an interval of length 2, namely, the interval $[j-2, j]$ within which the sequence of $2/\delta^{\text{tiny}}$ value-blocks lie. If, in this interval, the number of these value-blocks labelled $A_+$ exceeds the number labelled $A_-$ by at least $p^{\text{large}}$ (recall that $p^{\text{large}}$ is a large polynomial which is however polynomially smaller than $p^{\text{huge}}$), then (in any $\varepsilon$-approximate solution to $I_{CH}$, where, recall, $\varepsilon = \delta^{\text{tiny}}/10$), the cut $c(b_{i,j})$ in $I_{i,j}$ lies in either the right-hand or the left-hand value-block; otherwise, it lies in the central value-block. Note that these three possible positions may be converted to boolean values that influence circuit-encoder $C_i$.

The tunnel-boundary sensor agents use very similar bit detection gadgets to those of the sensor agents in their outputs (i.e., in their value-blocks in region $R_i$), but between their thin and dense value-blocks, they have an additional small value-block

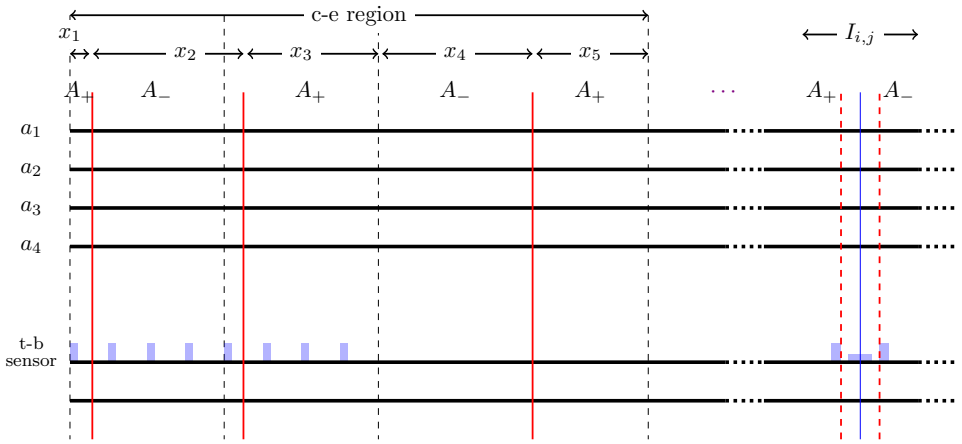


FIG. 7. *An example of how the tunnel-boundary sensor agents provide input to the circuit for their monitored intervals. Depending on the balance in labels for the value-blocks on the LHS, the bit-detection gadget of the tunnel-boundary sensor agent on the RHS assumes the leftmost, middle, or rightmost position. In this particular example, the number of value-blocks on the LHS for each label is balanced, and therefore the cut on the RHS (shown in blue) intersects the middle value-block of the bit-detection gadget. In this case, the tunnel-boundary sensor is not active. The other two possible positions for the cut on the RHS, when the tunnel-boundary sensor is active, are depicted by dashed red lines.*

(the block of value $9\kappa/10$ in Definition 4.6). This is because the tunnel-boundary sensor agents need to be able to assume three states: "excess of $A_+$," "excess of $A_-$," and "(approximately) balanced labels." The latter option corresponds to the cut associated with the tunnel-boundary sensor agent intersecting the middle value-block (therefore not jumping to either side), whereas the other two options correspond to the cut jumping to either the right or the left side, where the choice depends on the overrepresented label and the parity of the index of the tunnel-boundary sensor agent. It is straightforward (as before) to interpret these positions as boolean values. The main idea is that if the tunnel-boundary sensor agents are active (Definition 4.7 below), then this information will override the circuit $C_{VT}$ and generate an imbalance of labels in the feedback provided to the coordinate-restricting agents directly, essentially bypassing the output gates of $C_{VT}$. Since we have many tunnel-boundary sensor agents, however, extra care must be taken to make sure that no artificial solutions are introduced when coloring the domain. The details on how the input from the tunnel-boundary sensors affects the feedback to the coordinate-restricting agents are presented in subsection 5.1.3.

DEFINITION 4.7 (active tunnel-boundary sensor). *We say that tunnel-boundary sensor $b_{i,j}$ is* active *if $b_{i,j}$ in fact observes a sufficiently large label discrepancy in the c-e region, that $c(b_{i,j})$ lies in one of the two outer positions, left or right, and not in the central position. In particular, following Definition* 4.6*, a tunnel-boundary sensor agent will be active when at least $p^{\text{huge}}/n + p^{\text{large}}$ value-blocks of volume $\frac{1}{10} \cdot \frac{\delta^{\text{tiny}}}{2}$ in an interval monitored by the tunnel-boundary sensor agent receive the same label. We say that $b_{i,j}$ is* active toward $A_+$ *if $A_+$ is the overrepresented label, with similar terminology for $A_-$.*

*When tunnel-boundary sensor agent $b_{i,j}$ is active, it provides input to $C_i$ that causes $C_i$ to label the value held by $a_j$ and controlled by $C_i$, to be either $A_+$ or $A_-$;*

*the choice depends on the overrepresented label in* $[j-2, j]$ *and the parity of the index of the tunnel-boundary sensor agent. The precise feedback mechanism to the c-r agent* $a_j$ *by the tunnel-boundary sensor* $b_{i,j}$ *is described in subsection* 5.1.3.

When no tunnel-boundary sensors are active, the sequence of c-e cuts encodes a point in the Significant Region (Definition 4.14).

The advantage of using a polynomial sequence of value-blocks is that we can argue that in all but at most $n$ circuit-encoders, the tunnel-boundary sensor agents have value-blocks that are not cut by the c-e cuts, so we can be precise about how big of a disparity between blocks labelled $A_+$ and $A_-$ causes a tunnel-boundary sensor to be active, and for at most $n$ circuit-encoders, we regard them as having unreliable inputs (see Definition 4.10 and Observation 4.11).

*The gate agents.* In this subsection, we will design the agents that will be responsible for encoding the labelling circuit $C_{VT}$ of New Variant High-D Tucker. These agents will eventually provide feedback (in terms of a discrepancy of labels $A_+$ and $A_-$) to the coordinate-restricting agents (see subsection 5.1.2 for the details on the feedback mechanism). We start with the definition of the boolean gate gadgets, developed originally in [37].

*Boolean gate gadgets.* Consider a boolean gate that is an AND, an OR, or a NOT gate, denoted $g_\wedge$, $g_\vee$, and $g_\neg$, respectively. Let $in_1$, $in_2$, and $out$ be intervals such that $|in_1| = |in_2| = |out| = 1$. We will encode these gates using the following gate-gadgets.

$$g_\neg(in_1, out) = \begin{cases} 0.25 & \text{if } t \in in_1, \\ 7.5 & \text{if } t \in [\ell(out), \ell(out) + 1/20], \\ 7.5 & \text{if } t \in [r(out) - 1/20, r(out)], \\ 0 & \text{otherwise,} \end{cases}$$

$$g_\vee(in_1, in_2, out) = \begin{cases} 0.125 & \text{if } t \in in_1 \cup in_2, \\ 6.25 & \text{if } t \in [\ell(out), \ell(out) + 1/20], \\ 8.75 & \text{if } t \in [r(out) - 1/20, r(out)], \\ 0 & \text{otherwise,} \end{cases}$$

$$g_\wedge(in_1, in_2, out) = \begin{cases} 0.125 & \text{if } t \in in_1 \cup in_2, \\ 8.75 & \text{if } t \in [\ell(out), \ell(out) + 1/20], \\ 6.25 & \text{if } t \in [r(out) - 1/20, r(out)], \\ 0 & \text{otherwise.} \end{cases}$$

Note that the gadget corresponding to the NOT gate only has one input, whereas the gadgets for the AND and OR gates have two inputs. In the interval *out*, each gadget has two bit detection gadgets—in the case of the NOT gate these are even, but in the case of the AND and OR gates, they are uneven (see Figure 8). Also note that for the inputs, as well as the output of the NOT gate, the label on the LHS of the cut is $A_+$ and the label on the RHS will be $A_-$, whereas for the outputs to the OR and AND gate, the label on the LHS of the cut is $A_-$ and the label on the RHS is $A_+$. This can be achieved with the appropriate use of parity gadgets.

*Observation* 4.8. The boolean gate gadgets described above encode valid boolean NOT, OR, and AND operations.

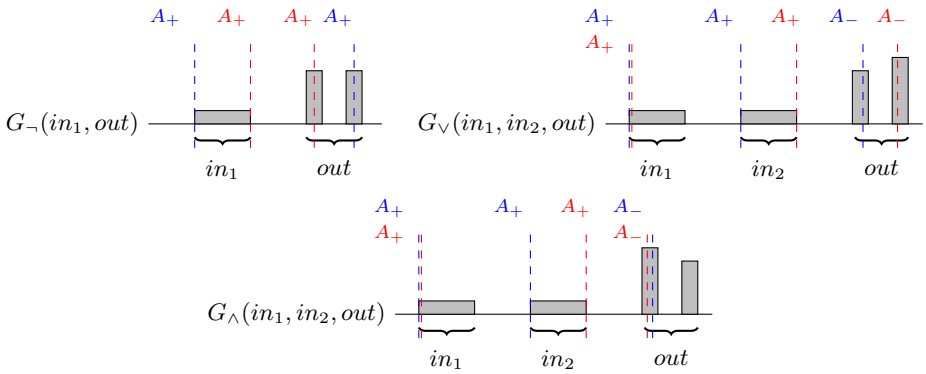


FIG. 8. *The boolean gate gadgets encoding the NOT, OR, and AND gates. For visibility, the valuation blocks are not according to scale. For the NOT gate, the input has value* 0.25 *and the output blocks have volume* 0.375 *each. For the OR (respectively, AND) gate, the input blocks have value* 0.125 *each and the output blocks have value* 0.3125 *and* 0.4375 *(respectively,* 0.4375 *and* 0.3125*). The cuts corresponding to pairs or triples of inputs and outputs have the same color, and the labels on the left-side of these cuts are shown and color-coded in the same way. For the NOT gate, when the input cut sits of the left (the blue cut), then the output cut must sit on the right (the blue cut), to compensate for the excess of $A_-$ and oppositely for when the input cut sits of the right (the red cut). For the OR and AND gates, again the cuts corresponding to two inputs and one output have the same color. For the OR gate, when both input cuts sit on the left (the blue cuts), the output cut sits on the left as well, to compensate for the excess of $A_-$ (notice that the LHS of the output cut is labelled $A_-$). When one input sits on the left and the other one on the right, the inputs detect no discrepancy in the balance of labels and the output jumps to the right, because the output blocks are uneven (the red cuts). The operation of the AND gate is very similar; here the cases shown are those where the inputs are* 0 *and* 0 *and the ouput is* 0 *(the blue cuts) and where the inputs are* 0 *and* 1 *and the output is still* 0 *(the red cuts).*

*Proof.* These gadgets encode the boolean gate operations in the following way: We will interpret the position of a cut $c$ relatively to $\ell(in_1)$, $\ell(in_2)$, and $\ell(out)$ as the input or the output to the gates, respectively. Specifically, for $j \in \{1, 2\}$, if $c \in [\ell(in_j), \ell(in_j) + \varepsilon]$, the input will be 0, and if $c \in [r(in_j) - \varepsilon, r(in_j)]$, the input will be 1. Similarly, if $c \in [\ell(out), \ell(out) + 1/20]$, the output will be 0 and if $c \in [r(out) - 1/20, r(out)]$, the output will be 1. If the inputs or the outputs lie on any other point in the corresponding intervals, the gate inputs and outputs are undefined, but it will be enforced by our construction that in a solution to $I_{CH}$, this will never happen.

For $g_\neg$, let's assume that the cut in $in_1$ lies in $[\ell(in_1), \ell(in_1)+\varepsilon]$, which means that a total value of approximately 0.25 is assigned to $A_-$ in the interval $in_1$ (recall that all cuts have an $A_+$ label on their LHS). To compensate, since the agent only has further value in *out*, the cut would have to lie in $[r(out) - 1/20, r(out)]$, and therefore by the interpretation of the inputs above, we can see that when the input is 0, the output is 1 and the gate constraint is satisfied. If the cut in $in_1$ lies in $[r(in_1) - \varepsilon, r(in_1)]$, then the value in the interval $in_1$ has been labelled $A_+$, and for the same reason, the cut in *out* has to lie in $[\ell(out), \ell(out) + 1/20]$, which encodes the case when the input is 1 and the output is 0. The arguments for the $g_\vee$ and $g_\wedge$ gadgets encoding the OR and AND gates, respectively, are very similar (noting that the cut intersecting *out* will have $A_-$ on its LHS). See Figure 8 for an illustration. ∎

Parity gadgets. A parity gadget is an agent $\alpha_{par}$ that has a single valuation block (i.e., an interval where the agent has a constant, nonzero value) of sufficiently small height and width, in a region between two such distinct valuation blocks of some other agent or agents (where we need the parity switch to take place), and furthermore, no other agent has any value in that interval. Since we are only allowed to use $n$ cuts, in a solution to $I_{CH}$, only $\alpha_{par}$'s cut $c(\alpha_{par})$ may lie in the region and therefore intersect this valuation block; obviously the cut has to lie close to the midpoint of the valuation block interval and it will switch the parity of the cut sequence. Throughout the reduction, we will not explain how to explicitly place the parity gadgets in the instance of $I_{CH}$ but rather we will assume without loss of generality that the LHSs of the cuts are labelled $A_+$, unless stated otherwise.

*Implementing the circuit using the gate gadgets.* For the circuits, at a high level, we will simulate the gates by gate agents, using the boolean gate gadgetry presented above. In particular, for any two-input gate, $g$ of the circuit with inputs $in_1, in_2$, agent $\alpha^g$ will have a bit detection value gadget that will encode the output of the gate, and furthermore, it will have value in some intervals $\mathcal{R}_k$ and $\mathcal{R}_\ell$, where the values of $in_1$ and $in_2$ lie, respectively, where $in_1$ and $in_1$ can either be the outputs of some gates $g_1$, $g_2$ of some previous level, or the outputs of the sensor agents, if $g$ is an input gate of $C_{VT}$. For an illustration, see Figure 9. The case of $g$ being a single-input gate is similar. The construction will make sure that agent $\alpha^g$ will only be satisfied with the consensus-halving solution if the gate constraint is satisfied.

Concretely, we will use the gate gadgets that will encode the gates of the circuit. For each gate of the circuit-encoder $C_1$ (see also subsection 5.1), we will associate a gate agent $\alpha_1^g, \ldots, \alpha_{|C_1|}^g$ with valuation given by the gadget

$$\mu_{a_i^g}(t) = \begin{cases} g_T(in_1^i, in_2^i, out^i) & \text{if } T \in \{\vee, \wedge\}, \\ g_T(in_1^i, out^i) & \text{if } T = \neg, \end{cases}$$

where $in_1^i$, $in_2^i$, and $out^i$ are nonoverlapping intervals that depend on whether $a_i^g$ corresponds to an input gate, an output gate, or an intermediate gate of the circuit. In particular, the following hold:

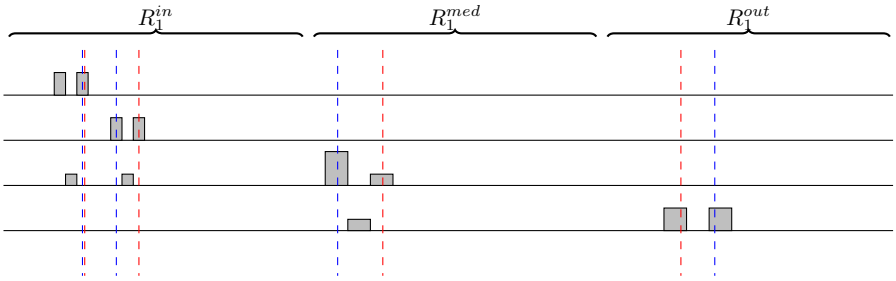


FIG. 9. *The basic idea behind the gate-agents encoding the gates of* $C_1$. *The picture denotes a simplified case where two input bits from the sensor agents are supplied to an enconding of an input AND gate of* $C_i$ *and the output bit of this gate is in turn supplied to the encoding of a NOT output gate of* $C_i$. *Note that if, for example, the sensor agents detect the values* 1 *and* 0, *respectively (the blue cuts), then the output of the AND gate is* 0 *(i.e., the blue cut sits at the left of the AND gate agent's bit detector) and the output of the circuit is* 1 *(again, see the blue cut that sits on the rightmost valuation block of the NOT gate agent). Similarly, if the sensor agents detect values* 1 *and* 1 *(the red cuts), then the output of the AND gate is* 1 *and the output of the circuit is* 0.

- If $a_i^g$ corresponds to an input gate of the circuit, $in_1^i$ and $in_2^i$ are the intervals in $R_i$ from which the outcomes of the sensor agents are read (via the position of the cuts); see the RHS of Figure 6. $out^i$ is another interval of $R_i$, which will be the same interval from which the input of the gate of the next level of the circuit will be read; see Figure 9.
- If $a_i^g$ corresponds to an intermediate gate of the circuit, then $in_1^i$ and $in_2^i$ are the intervals corresponding to the outputs of the gates of the previous level. We can assume without loss of generality that the labelling circuit operates with gates of fan-in at most 2, as any circuit can be converted to such a circuit at the expense of a logarithmic increase in depth. Again, $out^i$ will correspond to another interval, which will also be an input interval for a gate of the next level.
- If $a_i^g$ corresponds to an output gate of the circuit, then again $in_1^i$ and $in_2^i$ are the intervals corresponding to the outputs of the gates of the previous level, but $out^i$ will be intervals for which the coordinate-restricting agents will have positive value. In fact, the circuit will have $2n$ output gates, which will label value-blocks of the corresponding coordinate-restricting agent (two for each such agent) by $A_+$ or $A_-$ (assuming that the circuit operates as intended), depending on the label in $\pm[n]$ of the point in the domain "read" by the sensor agents $\mathcal{S}_i$. The details of the feedback mechanism are postponed for subsection 5.1.2.

We remark here that we have abstracted a few details regarding the design of the circuit-encoder $C_1$, when stating that the gate agents actually implement the circuit $C_{VT}$ of New Variant High-D Tucker. A more precise statement would be that it implements the circuit $C_{VT}$, together with a preprocessing circuit which performs the coordinate-transformation mentioned in the high-level overview of section 2, as well as a simple XOR circuit, which takes the exclusive-or of the output of $C_{VT}$ and the first sensor agent (the *reference* agent) $s_{1,1}$. The latter operation is necessary for the disorientation of the domain, which allows us to identify points when moving cuts from one endpoint of the c-e region to the other. See subsection 5.1 for more details.

**4.2. Features of solutions.** The main result of this section is Proposition 4.15, that in a solution to approximate Consensus-Halving as constructed here, the sequence of cuts in the c-e region are "evenly spaced" in the sense that the gap between consecutive cuts differs from 1 by at most an inverse-polynomial.

*Observation* 4.9 (at most $n$ cuts in the c-e region). Given an instance $I_{CH}$ derived by our reduction from an instance of New Variant High-D Tucker in $n$ dimensions, any inverse-polynomial approximate solution of $I_{CH}$ has the property that at most $n$ cuts lie in the coordinate-encoding region.

This is because the following two properties hold:

- every agent besides the coordinate-restricting agent has at least 9/10 of their value strictly to the right of the c-e region, and
- the value blocks of those agents do not overlap or overlap "minimally" for more than a single such agent to be "satisfied" by a single cut.

In particular, the construction ensures that for any agent besides the coordinate-restricting agent, this agent requires a *distinct* cut to balance her value in a solution, and therefore the cuts corresponding to those agents cuts cannot lie in the c-e region.

DEFINITION 4.10 (reliable input). *We will say that a circuit-encoder* receives reliable input *if no coordinate-encoding cut passes through value-blocks of its sensor agents.*

*Observation* 4.11. At most $n$ circuit-encoders fail to receive reliable input (by Observation 4.9 and the fact that sensors of distinct circuit-encoders have value in distinct intervals).

When a circuit-encoder receives reliable input, it is straightforward to interpret the labels allocated to its sensors, as boolean values, and simulate a circuit computation on those values, ultimately passing feedback to the coordinate-restricting agents via value-blocks that get labelled according to the output gates of the circuit being simulated. This is done via the gate agents and the use of the appropriately designed gate gadgets of the previous section.

DEFINITION 4.12. The Möbius-simplex. *The* Möbius-simplex in $n$ dimensions *consists of points* $\mathbf{x}$ *in* $\mathbb{R}^{n+1}$ *whose coordinates are nonnegative and sum to* 1. *We identify every point* $(x_1, \ldots, x_n, 0)$ *with the point* $(0, x_1, \ldots, x_n)$ *for all nonnegative* $x_1, \ldots, x_n$ *summing to* 1. *We use the following metric* $d(\cdot, \cdot)$ *on the Möbius-simplex, letting* $L_1$ *be the standard* $L_1$ *distance on vectors:*

$$d(\mathbf{x}, \mathbf{x}') = \min\Big(L_1(\mathbf{x}, \mathbf{x}'), \min_{\mathbf{z}, \mathbf{z}' : \mathbf{z} \equiv \mathbf{z}'} (L_1(\mathbf{x}, \mathbf{z}) + L_1(\mathbf{z}', \mathbf{x}'))\Big), \tag{4.1}$$

*where* $(0, x_1, \ldots, x_n) \equiv (x_1, \ldots, x_n, 0)$.

*How a consensus-halving solution encodes a point in the Möbius-simplex.* Let $I_{CH}$ be an instance of CONSENSUS-HALVING, obtained by reduction from NEW VARIANT HIGH-D TUCKER in $n$ dimensions, hence having c-e region $[0, n]$. Note that, by Observation 4.9, at most $n$ cuts may lie in the c-e region. A set of $k \leq n$ cuts of the coordinate-encoding region splits it into $k + 1$ pieces. We associate such a split with a point $\mathbf{x}$ in $\mathbb{R}^{n+1}$ as follows. The first coordinate is the distance from the LHS of the consensus-halving domain to the first cut, divided by $n$, the length of the c-e region. For $2 \leq i \leq k + 1$, the $i$th coordinate of $\mathbf{x}$ is the distance between the $i - 1$st and $i$th cuts, divided by $n$. Remaining coordinates are 0.

If there are $n - 1$ cuts in the c-e region, suppose we add a cut at either the LHS or the RHS. These two alternative choices correspond to a pair of points that have been identified as the same point, as described in Definition 4.12. (Observation 5.5 makes a similar point regarding transformed coordinates.)

*Observation* 4.13. Each circuit-encoder reads an "input" representing a point in the Möbius-simplex (as specified by the position of the cuts in the c-e region). Any circuit-encoder $C_i$ ($i \in [p^C]$) behaves like $C_1$ on a point $\mathbf{x}_i$, for which (for all $i, j \in [p^C]$) $d(\mathbf{x}_i, \mathbf{x}_j) \leq \delta^{\text{tiny}}$ (recall $d$ is defined in 4.1). Consequently, their collective output (the split between $A_+$ and $A_-$ of the value held by the coordinate-restricting agents) is the output of a single circuit-encoder averaged over a collection of $p^C$ points in the Möbius-simplex, all within $\delta^{\text{tiny}}$ of each other.

This follows by inspection of the way the $p^C$ circuit-encoders differ from each other: their sensor-agents are shifted but their internal circuitry is the same.

DEFINITION 4.14. The Significant Region of the Möbius-simplex $D$. *The* Significant Region *of* $D$ *consists of all points in* $D$ *where no tunnel-boundary sensors are active (where "tunnel-boundary sensors" and "active" are defined in Definition* 4.7*).*

*Remarks.* Looking ahead, certain points in the Significant Region encode NEW VARIANT HIGH-D TUCKER (namely, the ones in the "twisted tunnel," Definition 5.7).

The Significant Region contains the twisted tunnel, being a somewhat wider one-dimensional "tunnel" of inverse-polynomial width at most $1/p_w(n)$, whose central axis is the set of points $(\alpha, 1/n, \ldots, 1/n, 1/n - \alpha)$, where the endpoints are identified together (noting Definition 4.12). Topologically, the Significant Region is a high-dimensional Möbius strip.

PROPOSITION 4.15. *There is an inverse-polynomial value $\delta^w$ such that all points $\mathbf{x} = (x_1, \ldots, x_{n+1})$ in the Significant Region have coordinates $x_i$ that for $2 \le i \le n$ differ from $1/n$ by at most $\delta^w$ if $\mathbf{x}$ is encoded by the c-e cuts of an $\varepsilon$-approximate solution to one of our instances of* CONSENSUS-HALVING. *(Recall that $\varepsilon = \delta^{\text{tiny}}/10$.)*

*Thus, if an instance $I_{CH}$ of* CONSENSUS-HALVING *(obtained using our reduction) has a solution $S_{CH}$, then all the c-e cuts in $S_{CH}$ have the property that the distance between two consecutive c-e cuts differs from 1 by at most some inverse-polynomial amount.*

Before we proceed with the proof of the proposition, we will state a few simple lemmas that will be used throughout the proof. The following definitions will be useful later.

DEFINITION 4.16 (cut $\delta$-close to integer point). *For $\ell \in \{0, \ldots, n\}$, we will say that a cut $c$ is $\delta$-*close to integer point *$\ell$ if it lies in $[\ell - \delta, \ell + \delta]$. We will say that cut $c$ is $\delta$-*close to an integer point *if there is some integer $\ell \in \{0, \ldots, n\}$ such that $c$ is $\delta$-close to integer point $\ell$.*

DEFINITION 4.17 (monochromatic interval of label $A_j$). *An interval $I$ is called a* monochromatic interval *if it is not intersected by any cuts (thus it receives a single label). If for $A_j \in \{A_+, A_-\}$, $I$ is labelled with $A_j$, then we will say that $I$ is a* monochromatic interval of label $A_j$.

By Definition 4.6, a tunnel-boundary sensor agent will be active when a large enough fraction of the interval that it monitors receives the same label. In more detail, the valuation of a tunnel-boundary sensor agent in the c-e region consists of a set of $2p^{\text{huge}}/n$ value-blocks (of volume $\frac{1}{10} \cdot \frac{\delta^{\text{tiny}}}{2}$ each; see Figure 7), and it will be active when at least $p^{\text{huge}}/n + p^{\text{large}}$ of those blocks receive the same label. What this means for a monitored interval $[j-2, j]$ is that its tunnel-boundary sensor agent will be active if subintervals of length $1 + np^{\text{large}}/p^{\text{huge}}$ get the same label, and equivalently if subintervals of length $1 + \delta^{\text{tiny}} p^{\text{large}}$ get the same label. Let

$$\delta = n\delta^{\text{tiny}} p^{\text{large}}.$$

Based on the relative sizes of $\delta^{\text{tiny}}$ and $p^{\text{large}}$, we can assume $\delta < \frac{1}{2n}$. With this definition of $\delta$, if any monitored interval $[j-2, j]$ has a larger than $1 + \delta/n$ monochromatic subinterval, then tunnel-boundary sensor agents $b_{i,j}$ are active.

LEMMA 4.18. *For $\ell \in \{0, n-k\}$ with $k > 1$, consider the interval $I = [\ell, \ell + k]$ of length $k$ and suppose that there are at most $k-2$ cuts in this interval. Then for any $i \in p^C$, at least one of the tunnel-boundary sensors $b_{i,\ell+2}, \ldots, b_{i,\ell+k}$ monitoring the subintervals in $I$ will be active.*

*Proof.* In $I$, there are at least $k-1$ intervals monitored by tunnel-boundary sensor agents and we only have at most $k-2$ cuts at our disposal. With $k-2$ cuts, we can partition an interval of length $k$ in at most $k-1$ intervals, the largest of which, call it $I_{\max}$, will have length at least $1 + 1/k$. Since $\delta < 1/2n$, the length of $I_{\max}$ is actually larger than $1 + \delta$. The lemma follows then from the fact that, from the way that the monitored intervals cover the interval $I$, $I_{\max}$ will contain a monochromatic interval

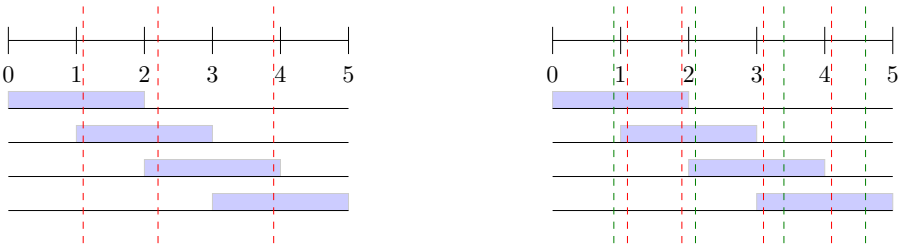


FIG. 10. *The case of an interval of length $k$ being intersected by $k-2$ (left) or $k-1$ (right) cuts, here $k=5$. The monitored subintervals are depicted in blue. On the left, an interval of length 5 is cut by only three cuts. The interval defined by the second and third cuts is of length larger than $1+1/k=6/5$. On the right, an interval of length 5 is cut by four cuts. It is possible to achieve an approximately balanced partition, but only if all cuts are $\delta$-close to integer coordinates and specifically to midpoints of the monitored subintervals, which is indicated by the red cuts. A case where this does not happen is indicated by the green cuts, where the tunnel-boundary sensor agent of interval $[2,4]$ is active. Note that while in the figure, in both cases, the interval of length $k$ contains only full monitored intervals, the same arguments go through if it contains half intervals instead, e.g., considering the interval $[1,5]$ and 2 cuts (left) and three cuts (right).*

of length at least $1+\delta$, which will be entirely contained in some monitored interval, and the corresponding tunnel-boundary sensor agent will be active. ∎

LEMMA 4.19. *For $\ell \in \{0, n-k\}$ with $k>1$, consider the interval $I=[\ell,\ell+k]$ of length $k$ and suppose that there are $k-1$ cuts in this interval. Recall that we let $\delta = n\delta^{\text{tiny}}p^{\text{large}}$. Then either*

- *each of the $k-1$ cuts in $I$ will be $\delta$-close to a different integer point and these integer points will be the midpoints of the monitored subintervals contained entirely in $I$ or*
- *for any $i \in p^C$, at least one of the tunnel-boundary sensors $b_{i,\ell+2},\dots,b_{i,\ell+k}$ monitoring the subintervals in $I$ will be active.*

*Proof.* Assume $\ell=0$ and we have the interval $[0,k]$; the other case is similar.

By the pigeonhole principle, some unit interval $[j-1,j]$ has no cuts (for $j\in[n]$). So the entirety of that interval gets a single label. Then a fraction at least $1-\delta/n$ of the adjacent unit intervals $[j-2,j-1]$ (for any $j>1$) and $[j,j+1]$ (for any $j<k$) must get the opposite label (if not, the tunnel-boundary sensor monitoring $[j-2,j]$ or $[j-1,j+1]$ will be active). Inductively, for an interval $[j',j'+1]$ with $j''=|j-j'|$, that interval must receive a fraction $1-j''\delta/n$ of the opposite label to its neighbors (to avoid activating a tunnel-boundary sensor). This requires all cuts to lie within $\delta$ of the integer points $1,\dots,k-1$, in order to achieve this alternation of labels. See Figure 10. ∎

We are now ready to proceed with the proof of Proposition 4.15.

*Proof of Proposition* 4.15. First, recall that by Observation 4.9, at most $n$ cuts can lie in the c-e region. Also recall that from Definition 4.6, for the circuit encoder $C_1$, the tunnel-boundary sensor agent $b_{1,j}$, $j\in\{2,\dots,n\}$ has valuation only in the interval $[j-2,j]$ of the c-e region, i.e., it monitors the interval $[j-2,j]$. The tunnel-boundary sensor agent $b_{i,j}$ for $i\in\{2,\dots,p^C\}$ is a $(i-1)\frac{\delta^{\text{tiny}}}{p^C}$-shifted version of $b_{1,j}$. We will make the argument for the tunnel-boundary sensor agents of the circuit-encoder $C_1$; the argument for any $C_i$, with $i\neq 1$ is very similar.

It suffices to prove that if consecutive cuts are too far apart or too close together, some tunnel-boundary sensor agent will be active. In what follows, again let $\delta = n\delta^{\text{tiny}}p^{\text{large}}$, which as noted earlier is an inverse-polynomial in $n$.

*Case* 1*: The cuts are too far apart.* First, consider the case when two consecutive cuts are too far apart (by more than 1 plus some inverse-polynomial amount $2\delta$). More formally, assume that there are two cuts $c_1$ and $c_2$ such that $c_2 > c_1$ and $c_2 - c_1 > 1 + 2\delta$. Then, as we explain below, there is some $j \in \{2, \ldots, n\}$ such that some subinterval $I_j = [j_1, j_2] \subseteq [j-2, j]$ with $j_2 - j_1 > 1 + \delta$ will receive a single label, either $A_+$ or $A_-$. In particular, we have the following cases:

- There is a $j$ such that $[c_1, c_2] \subseteq [j-2, j]$. In that case, $[c_1, c_2]$ is such a monochromatic subinterval.
- There is a $j$ such that $[j-2, j] \subseteq [c_1, c_2]$. In that case, the whole monitored subinterval $[j-2, j]$ is such a monochromatic subinterval.
- For all $j$, the interval $[j-2, j]$ is intersected by at most one cut $c_\ell$, $\ell \in \{1, 2\}$. Obviously, both cuts will intersect some interval, since they lie in the c-e region. Consider cut $c_1$ and let $[j-2, j]$ be an interval that is intersected by $c_1$. If $c_1$ lies in $[j-2, j-1]$, then, since there exists no other cut between $c_1$ and $c_2$ and since $c_2$ does not intersect $[j-2, j]$ by assumption, the interval $[c_1, j]$ will be a monochromatic interval of length at least $1 + \delta$ and we are done. If $c_1$ lies in $[j-1, j]$, then first observe that $j \neq n$, as otherwise both cuts $c_1$ and $c_2$ would have to lie in $[n-2, n]$, violating the assumption of the case. Therefore, we can look at the interval $[j-1, j+1]$ and notice that again by the assumption of the case, since cut $c_1$ does intersect the interval $[j-1, j+1]$, we must have that $c_2 > j+1$. This is either impossible (when $j = n-1$) or otherwise $[c_1, j+1]$ is a monochromatic interval of length at least $1 + \delta$, and we are done.

*Case* 2*: The cuts are too close together.* Now consider the case when two consecutive cuts are too close together, closer than $1 - 2n\delta$. More formally, assume that there are two consecutive cuts $c_1$ and $c_2$ in the c-e region such that $c_2 > c_1$ and $c_2 - c_1 < 1 - 2n\delta$. Since the cuts are close together, there exists a monitored interval that is intersected by both $c_1$ and $c_2$ and let $[j-2, j]$ be such an interval. Notice that if there exists no other cut that intersects $[j-2, j]$, then $[j-2, c_1] \cup [c_2, j]$ is a union of subintervals of length at least $1+\delta$ that receive the same label, and we are done. Therefore, there must exist at least three cuts that lie in $[j-2, j]$. We consider three cases. *There are five or more cuts in* $[j-2, j]$. This is an easy case to argue, as if that happens, there will be some interval, either $[0, j-2]$ or $[j, n]$ of length $k$, that is only intersected by at most $k-2$ cuts. By Lemma 4.18, some tunnel-boundary sensor agent will be active and we are done.

*There are four cuts in* $[j-2, j]$. Consider the intervals $[0, j-2]$ and $[j, n]$. If either $[0, j-2]$ is intersected by at most $(j-2)-2$ cuts or $[j, n]$ is intersected by at most $n-j-2$ cuts, then by Lemma 4.18, some tunnel-boundary sensor will be active and we are done. Note also for completeness that if $j = 2$ (respectively, $j = n$), it is necessarily the case that $[j, n]$ (respectively, $[0, j-2]$) is intersected by $n-4$ cuts and Lemma 4.18 again applies. Therefore, we can assume that $j \in \{3, \ldots, n-1\}$ and that there are exactly $(j-2)-1$ cuts in $[0, j-2]$ and $n-j-1$ cuts in $[j, n]$.

Consider the interval $[j, n]$ without loss of generality, as the argument for $[0, j-2]$ is symmetric. By Lemma 4.19, we know that the cuts in $[j, n]$ are $\delta$-close to integer points and particularly, they are $\delta$-close to the midpoints of the monitored intervals $[j, j+2], \ldots, [n-2, n]$. This implies that in the monitored subinterval $[j, j+2]$, the subinterval $[j, j+1-\delta]$ will be a monochromatic interval of label $A_j$ for some $A_j \in \{A_+, A_-\}$,

In turn, this implies that $[j-1, j]$ has a monochromatic subinterval of length at least $1 - \delta$ that receives the label $A_{-j}$, where $A_{-j} \in \{A_+, A_-\}$ is the complementary label to $A_j$, for the tunnel-boundary sensor agent to not be activated, which is only

possible if one of the four cuts in $[j-2, j]$ is $\delta$-close to the integer point $j$. Propagating the effect of this cut sequence/labelling into the monitored interval $[j-2, j]$ in question, we obtain that $[j-2, j-1]$ also contains a monochromatic interval of length at least $1-\delta$ and label $A_j$, as otherwise tunnel-boundary sensor agent $b_{1,j}$ would be active. From this discussion, it follows that

- all the cuts in $[j-2, j]$ are $\delta$-close to integer coordinates within the interval and
- there is at least one cut in $[j-2, j]$ that is $\delta$-close to the midpoint $j-1$ of the monitored interval, one cut that is $\delta$-close to the right endpoint $j$ of the monitored interval, and at least one cut that is $\delta$-close to the left endpoint $j-2$ of the monitored interval,

where the very last statement follows from the symmetric argument to the one developed above for the interval $[0, j-2]$. See Figure 11 for an illustration.

Now, we consider three cases with respect to the positions of the four cuts in $[j-2, j]$, illustrated in Figure 12. From the discussion above, we know that three of the cuts will be $\delta$-close to the left-endpoint, midpoint, and right-endpoint of $[j-2, j]$, respectively, so it suffices to consider the cases depending on the position of the fourth cut. Henceforth, we use $c_1, c_2, c_3$ to denote these three cuts, from left to right in terms of their position within the interval and $\tilde{c}$ to denote the aforementioned fourth cut.

1. $\tilde{c}$ is $\delta$-close to $j-1$. In that case, assuming without loss of generality that $\tilde{c} < c_2$, due to the parity of the cut sequence, the union of intervals $[c_1, \tilde{c}] \cup [c_2, c_3]$ contains monochromatic intervals of the same label and length at least $1+\delta$, and therefore tunnel-boundary sensor $b_{1,j}$ will be active. See the LHS of Figure 12.
2. $\tilde{c}$ is $\delta$-close to $j$. In that case, it is possible that $[j-2, j]$ does not contain a union of monochromatic intervals of the same label of length at least $1+\delta$.

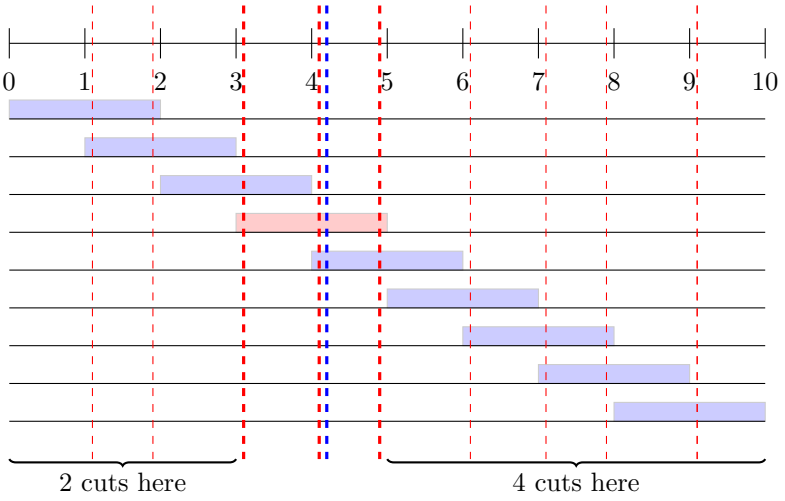


FIG. 11. *The case in which there are four cuts in the interval $[j-2, j]$ (shown in red), here for $j=5$. The three cuts that lie within distance $\delta$ from the left endpoint, midpoint, and right endpoint of $[j-2, j]$ are depicted with thick dashed red lines. The other cut in the interval (based on the position of which the different cases are considered) is depicted by a thick dashed blue line, and in this particular case, it is shown to be $\delta$-close to the midpoint of the interval. Notice that the positioning of the cuts in $[0, 3]$ and in $[5, 10]$ is such that the cuts are $\delta$-close to integer coordinates which are the midpoints of the monitored subintervals. If that was not the case, then some subintervals would be sufficiently imbalanced and the corresponding tunnel-boundary sensor agent would be active.*

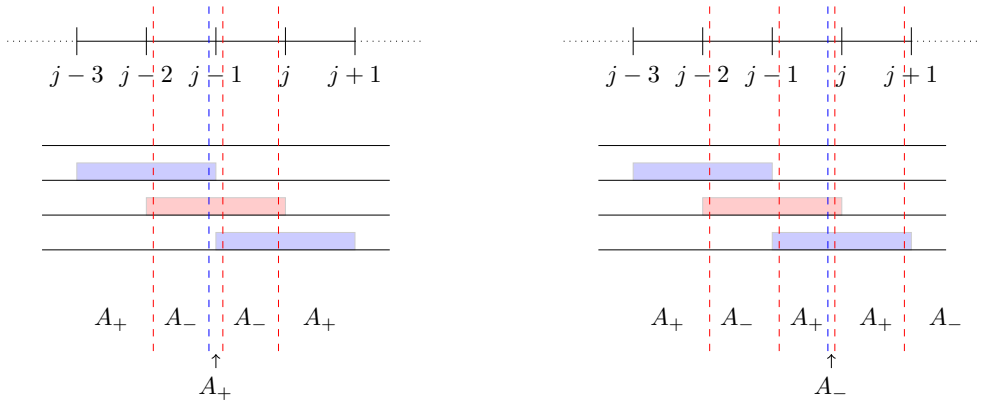


FIG. 12. *The two subcases of the case when there are four cuts in the interval $[j-2, j]$. The three cuts $c_1, c_2$, and $c_3$ that are $\delta$-close to the integer points $j-2$, $j-1$, and $j$ in the interval are shown in red; the other cut $\tilde{c}$ is shown in blue. On the left, when $\tilde{c}$ is $\delta$-close to the midpoint $j-1$ of the interval, most of $[j-2, j]$ is colored with the same label, here $A_-$, by the parity of the cut sequence. On the right, $\tilde{c}$ is $\delta$-close to the right endpoint $j$ of the interval, which means that, by the parity of the cut sequence, most of $[j-1, j+1]$ receives the label $A_+$, since if there is another cut in the interval, it is constrained by the arguments of the proof to be $\delta$-close to the right endpoint $j+1$ (shown in red here).*

However, by the parity of the cut sequence, in the interval $[j-1, j+1]$, now most of the interval $[j-1, j+1]$ will receive the same label, and $[j-1, j+1]$ will contain a union of monochromatic intervals of the same label of total length at least $1+\delta$, activating the tunnel-boundary sensor $b_{1,j+1}$. See the RHS of Figure 12.

3. $\tilde{c}$ is $\delta$-close to $j-2$. This case is symmetric to case 2 above.

*There are three cuts in* $[j-2, j]$. Again, considering the intervals $[0, j-2]$ and $[j, n]$ as we did in the case of four cuts in $[j-2, j]$, we can now observe that one of the intervals will be intersected by at most $k-1$ cuts, where $k \in \{j-2, n-j\}$ is its length. Furthermore, if it is intersected by fewer than $k-1$ cuts, by Lemma 4.18 some tunnel-boundary sensor agent will be active and we are done. Therefore, we will consider the case when one of the intervals is intersected by exactly $k-1$ cuts and let $[j, n]$ be that interval, without loss of generality, as the argument for $[0, j-2]$ is symmetric.

Following exactly the same arguments as in the second and third paragraph of the case of four cuts above, we can establish a very similar statement, namely, that

- all the cuts in $[j-2, j]$ are $\delta$-close to integer coordinates within the interval and
- there is at least one cut in $[j-2, j]$ that is $\delta$-close to the midpoint $j-1$ of the monitored interval and one cut that is $\delta$-close to the right endpoint $j$ of the monitored interval.

Again, letting $c_1$ and $c_2$ denote the two cuts mentioned in the second item above from left to right in terms of their positions, we will consider some cases depending on the position of the third cut, which we will denote by $\tilde{c}$.

1. $\tilde{c}$ is $\delta$-close to $j-2$. In that case, considering the intervals $[\tilde{c}, c_1]$ and $[c_1, c_2]$, we observe that since the cuts $\tilde{c}$, $c_1$, and $c_2$ are $\delta$-close to the integer points $j-2, j-1$, and $j$, respectively, both intervals have length at least $1-2\delta$. However, this contradicts the assumption of the case, namely, that there exists two cuts in $[j-2, j]$ that are within distance at most $1-n\delta$ from each other. See Figure 13, LHS.

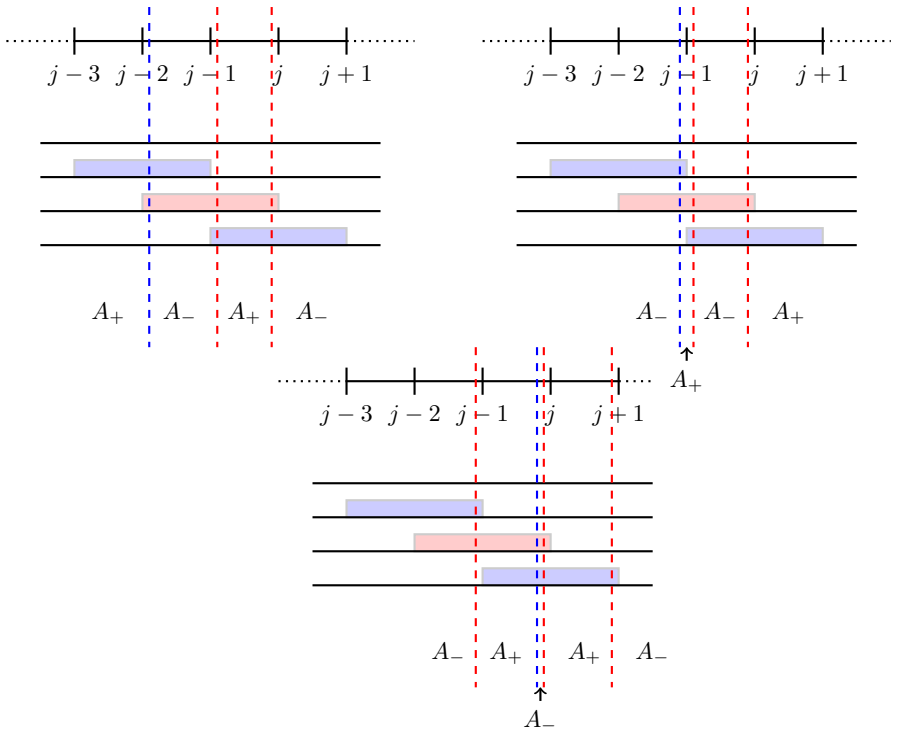


FIG. 13. *The three subcases of the case when there are three cuts in the interval* $[j-2, j]$. *The two cuts* $c1, c2$ *that are delta-close to the integer points* $j-1$ *and* $j$ *in the interval are shown in red, the other cut* $c$ *is shown in blue. On the left, when* $\tilde{c}$ *is* $\delta$*-close to the left endpoint* $j-2$ *of the interval, at least one of the subintervals defined by the cuts will have length at least* $1-2\delta$, *contradicting the assumption of the case. On the right,* $\tilde{c}$ *is* $\delta$*-close to the midpoint* $j-1$ *of the interval* $[j-2, j]$ *and by the parity of the cut sequence, most of the interval receives the same label, here* $A_-$. *Finally, at the bottom,* $\tilde{c}$ *is* $\delta$*-close to the right endpoint* $j$ *of the interval, which means that, by the parity of the cut sequence, most of* $[j-1, j+1]$ *receives the label* $A_+$, *since if there is another cut in the interval, it is constrained by the arguments of the proof to be* $\delta$*-close to the right endpoint* $j+1$ *(shown in red here).*

2. $\tilde{c}$ is $\delta$-close to $j-1$. In that case, similarly to case 1 for the case of four cuts, the parity of the cut sequence is such that most of $[j-2, j]$ will receive the same label and in particular $[j-2, j]$ will contain a union of monochromatic intervals of the same label with total length at least $1+\delta$, activating tunnel-boundary sensor $b_{1,j}$. See Figure 13, middle.
3. $\tilde{c}$ is $\delta$-close to $j$. Again, similarly to case 2 for the case of four cuts, it is possible that $[j-2, j]$ does not contain a union of monochromatic intervals of the same label of length at least $1+\delta$. However, by the parity of the cut sequence, in the interval $[j-1, j+1]$, now most of the interval $[j-1, j+1]$ will receive the same label, and $[j-1, j+1]$ will contain a union of monochromatic intervals of the same label of total length at least $1+\delta$, activating the tunnel-boundary sensor $b_{1,j+1}$. See Figure 13, RHS.

This completes the proof. ∎

**5. Reducing from** NEW VARIANT HIGH-D TUCKER **to** CONSENSUS-HALVING**.** In subsection 5.1 we give an overview of aspects of how we construct an instance $I_{CH}$ of $\varepsilon$-CONSENSUS-HALVING (in poly-time) from an instance of NEW VARIANT

HIGH-D TUCKER, for inverse polynomial $\varepsilon$. Subsection 5.2 describes the new coordinate system for the Möbius-simplex $D$ and establishes key properties. Subsection 5.3 presents a coloring function of $D$ in terms of the coordinate system of subsection 5.2. Subsection 5.4 describes how to construct a purported solution to $n$-dimensional NEW VARIANT HIGH-D TUCKER from a solution to $\varepsilon$-CONSENSUS-HALVING. In section 6 we prove that a solution to NEW VARIANT HIGH-D TUCKER that is obtained by reducing to $\varepsilon$-CONSENSUS-HALVING, solving it, and converting that solution to a solution to $n$-dimensional NEW VARIANT HIGH-D TUCKER really is a valid solution.

**5.1. Overview of the construction of an instance of the $\varepsilon$-CONSENSUS-HALVING problem from an instance of NEW VARIANT HIGH-D TUCKER.** We define the reduction from NEW VARIANT HIGH-D TUCKER (Definition 3.5) to $\varepsilon$-CONSENSUS-HALVING.

Let $I_{VT}$ be an instance of NEW VARIANT HIGH-D TUCKER in $n$ dimensions; let $C_{VT}$ be the boolean circuit that represents it. $I_{CH}$ will be the corresponding instance of CONSENSUS-HALVING. We list ingredients of $I_{CH}$ and give notation to represent them, as follows. $A$ is the consensus-halving domain, an interval of the form $[0, poly(n)]$. Any agent $a$ has a measure $\mu_a : A \longrightarrow \mathbb{R}$ represented as a step function (thus having a polynomial number of steps).

- $I_{CH}$ has $n$ *coordinate-restricting agents* $a_1, \ldots, a_n$ (Definition 4.2). See Figure 14.
- The consensus-halving domain $A$ of $I_{CH}$ has a *coordinate-encoding region* (c-e region) (Definition 4.1) consisting of the interval $[0, n]$.
- $I_{CH}$ has $p^C$ *circuit-encoders* $C_1, \ldots, C_{p^C}$ (see subsections 5.1.1 and 5.1.4).
  - Each $C_i$ has a set $\mathcal{A}_i$ of agents (see Figure 14) which includes $C_i$'s sensor agents, also circuit-encoding agents (below).
  - Each $C_i$ has an associated circuit-encoding region $R_i$ of $A$; each $R_i$ is an interval of polynomial length, and the $R_i$ do not intersect with each other or with the coordinate-encoding region.
  - $\mathcal{A}_i$ contains a polynomial number of *circuit-encoding agents* (one for each gate of $C_{VT}$), having value in $R_i$.

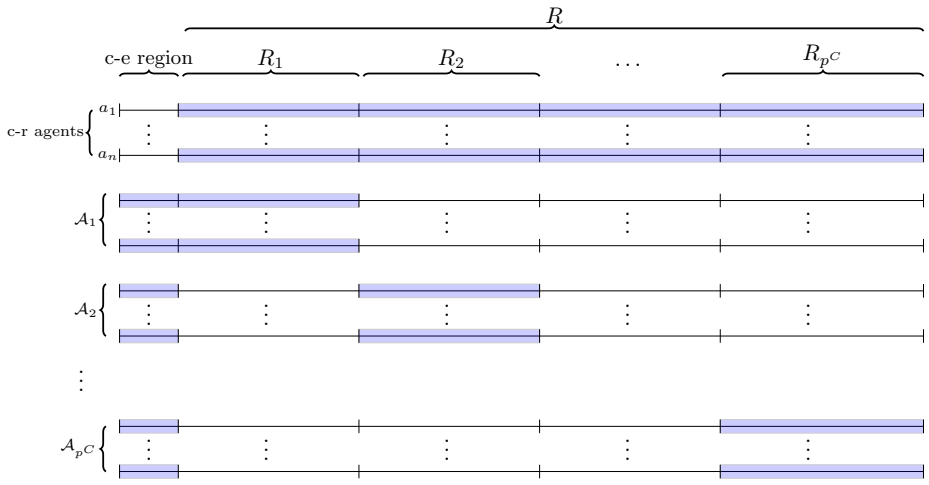


FIG. 14. *An overview of $I_{CH}$, denoting all the different regions and the agents of $C_1 \ldots, C_n$, as well as the coordinate-restricting agents. The highlighted areas denote that the corresponding agent has nonzero value on these regions.*

– Each $C_i$ has $p^{\text{huge}}$ *sensor agents* as defined in Definition 4.5, each of which has a block of value 1/10 in a small subinterval of the c-e region as specified in Definition 4.5, and further value in region $R_i$.
– Each $C_i$ has $n-1$ *tunnel-boundary sensor agents* as in Definition 4.6.

*Remarks.* We associate one cut with each agent; let $c(a)$ be the cut associated with agent $a$. The cuts $c(a)$ for coordinate-restricting agents are called the c-e cuts. A straightforward consequence of Proposition 4.15 is that in any solution, either all $n$, or $n-1$, of the coordinate-encoding cuts must lie in the coordinate-encoding region. All other cuts must lie in the regions $R_i$; indeed, every cut, other than the c-e cuts, is constrained by the value of its associated agent, to lie in a small interval that does not overlap any other such intervals. In the event that a c-e cut lies outside the c-e region, we refer to it as a "stray cut," and while such a cut may initially appear to interfere with the functioning of the circuitry, a "double-negative" lemma ensures that the duplication of the circuit using $p^C$ circuit-encoders allows the circuitry to be robust to this problem.

**5.1.1. Construction of $C_1$.** Recall $C_{VT}$ is the boolean circuit in the instance $I_{VT}$ of New Variant High-D Tucker.

- We assume that $C_{VT}$ has $2n$ output gates $g_1, \ldots, g_n$ and $g_{-1}, \ldots, g_{-n}$ having the property that exactly one of them will take value TRUE (this may be enforced syntactically). $g_i$ getting value 1 (TRUE) means that the point at coordinates represented by the input gets colored $i$.
- $C_{VT}$ has $n \cdot \text{polylog}(n)$ input gates, representing the coordinates of a point in $B = [-1, 1]^n$, each represented with inverse-polynomial precision.

We describe how circuit-encoder $C_1$ is derived from $C_{VT}$. The subsequent circuit-encoders can then be specified in terms of $C_1$. Each gate $g$ of $C_{VT}$ is simulated using a gate agent $a(g)$, as detailed in subsection 4.1, in the corresponding subsection. $a(g)$'s cut $c(a(g))$ occupies a right position, or a left position, representing TRUE or FALSE, as a function of the cut(s) that represent boolean inputs to $g$.

The circuit-encoding agents $\mathcal{A}_1$ of $C_1$ thus include $2n$ gate agents whose corresponding cuts simulate the values of the output gates of $C_{VT}$, provided that the input represented by the c-e cuts lies in the Significant Region (Definition 4.14). The positions of these cuts affect the labels of blocks of value held by the $n$ coordinate-restricting agents, as detailed in subsection 5.1.2.

Definition 5.1. Reference sensor-agent. *Noting from Definition* 4.5 *that the sensor agents for $C_i$ are denoted $\mathcal{S}_i = \{s_{i,1}, \ldots, s_{i,p^{\text{huge}}}\}$, we let $s_{1,1}$ be the* reference sensor-agent*: Outputs produced by the circuit $C_i$ are taken with reference to the value*[4] *$s_{1,1}$, in the sense that after simulating $C_{VT}$ we take the exclusive-or with $s_{1,1}$.*

This crucial technique of Definition 5.1 performs the task of disorienting the domain while at the same time ensuring continuity when we move a cut from the LHS of the c-e region to the RHS. Note that the XOR operation can be implemented using a rather simple subcircuit based on our boolean gate gadgets.

*Preprocessing, prior to simulating $C_{VT}$.* For each $C_i$, we take all $p^{\text{huge}}$ input bits, which appear in up to $n+1$ blocks of consecutive 1's and 0's, and convert them into the coordinates of a point in the Möbius-simplex (Definition 4.12). As noted earlier (Observation 4.11) at most $n$ circuit-encoders may receive ill-defined inputs caused by c-e cuts cutting through value-blocks in the c-e region that belong to their sensor agents; we simply assume that the output of those agents is unreliable, indeed adversarially chosen.

[4] We are using $s_{1,1}$ to denote the boolean value taken by $s_{1,1}$ as well as the sensor itself.

We then perform a coordinate transformation described in subsection 5.2. A subset of points in the Möbius-simplex maps to a copy of the domain $B$ of the instance $I_{VT}$ (recall Definition 3.5). These points get their coordinates passed directly to a copy of $C_{VT}$, and the outputs of $C_{VT}$ are used to provide feedback to the c-r agents as described in subsection 5.1.2 (and discussed in Observations 4.11 and 4.13). Other points get colored in a manner that avoids allowing bogus solutions to $I_{CH}$ (i.e., ones that do not encode solutions to $I_{VT}$).

**5.1.2. The output gates of $C_1$ and the feedback they provide to the coordinate-restricting agents.** $C_{VT}$ has output gates $g_j$, $j \in \pm[n]$, with the property that when inputs are well-defined, exactly one output gate evaluates to TRUE. A circuit-encoder simulates $C_{VT}$ using the gate gadgets introduced in subsection 4.1. Let $x_{REF} \in \{\text{TRUE}, \text{FALSE}\}$ be the negation of the value of the reference sensor (Definition 5.1). We use additional gates $g'_j$, $j \in \pm[n]$, where

- if $g_j = g_{-j} = \text{FALSE}$, then $g'_{|j|} = \text{TRUE}$ and $g'_{-|j|} = \text{FALSE}$;
- if $j > 0$ and $g_j = \text{TRUE}$ (so $g_{-j} = \text{FALSE}$), then $g'_j = g'_{-j} = \text{TRUE} \oplus x_{REF}$;
- if $j < 0$ and $g_j = \text{TRUE}$ (so $g_{-j} = \text{FALSE}$), then $g'_j = g'_{-j} = \text{FALSE} \oplus x_{REF}$.

Each of the c-r agents $a_1, \ldots, a_n$ has two value-blocks of value $1/(2p^C)$ in region $R_1$, and each gate $g'_j$ of $C_1$ is able to select the label of one of these value-blocks. (Recall that the boolean value at a gate is represented by two positions that may be taken by the corresponding cut, so that a block of value lies between these two positions.) Figure 15 shows an example of how this feedback works.

**5.1.3. How $C_1$'s tunnel-boundary sensors affect the feedback mechanism.** Let $A_j \in \{A_+, A_-\}$ and let $A_{-j} \in \{A_+, A_-\}$, $A_{-j} \neq A_j$ be the complementary label. The tunnel-boundary sensor agents $b_{1,2}, \ldots, b_{1,n}$ affect the output of the circuit-encoders as follows:

1. If none are active, the $2n$ outputs of $C_1$ are computed as described in subsection 5.1.2.
2. If $j$ is *odd* and $b_{1,j}$ is active in direction $A_j$, then the output gates $g'_j, g'_{-j}$ are both set to the value that causes c-r agent $a_j$ to observe more $A_j$.
3. If $j$ is *even* and $b_{1,j}$ is active in direction $A_j$, then the output gates $g'_j, g'_{-j}$ are both set to the value that causes c-r agent $a_j$ to observe more $A_{-j}$.

Rules 2 and 3 override rule 1, which allocates values that directly encode values output by $C_{VT}$. Note that the gadgetry of the circuit can ensure that either an excess of $A_+$ or an excess of $A_-$ can be shown to the corresponding c-r agent as feedback, as the circuit can convert the input value encoded by the value gadget of the tunnel-boundary sensor agent in $R_1$ to either a "right" or "left" output position, depending on the parity of the index. Also, if more than one tunnel-boundary sensor agent is active, they all affect their corresponding gates. The reason for requiring tunnel-boundary sensors of different parities to feedback different labels to the c-r agents is to be consistent with the definition of "consistent colours"; see Definition 6.8.

Note that we do not define the behavior of the tunnel-boundary sensor agents in terms of the reference sensor. They essentially look for an imbalance between $A_+$ and $A_-$ within some interval of length 2, and when they find a sufficiently large imbalance, they force the circuit $C_1$ to show their associated c-r agent more of the overrepresented or underrepresented label, depending on their parity. This is done via the bit-detection gadgets on their RHS, which, as explained in subsection 4.1, jump to the right or to the left, depending on which label is in excess, or remain in the middle if the two labels are approximately balanced; see Figure 7.

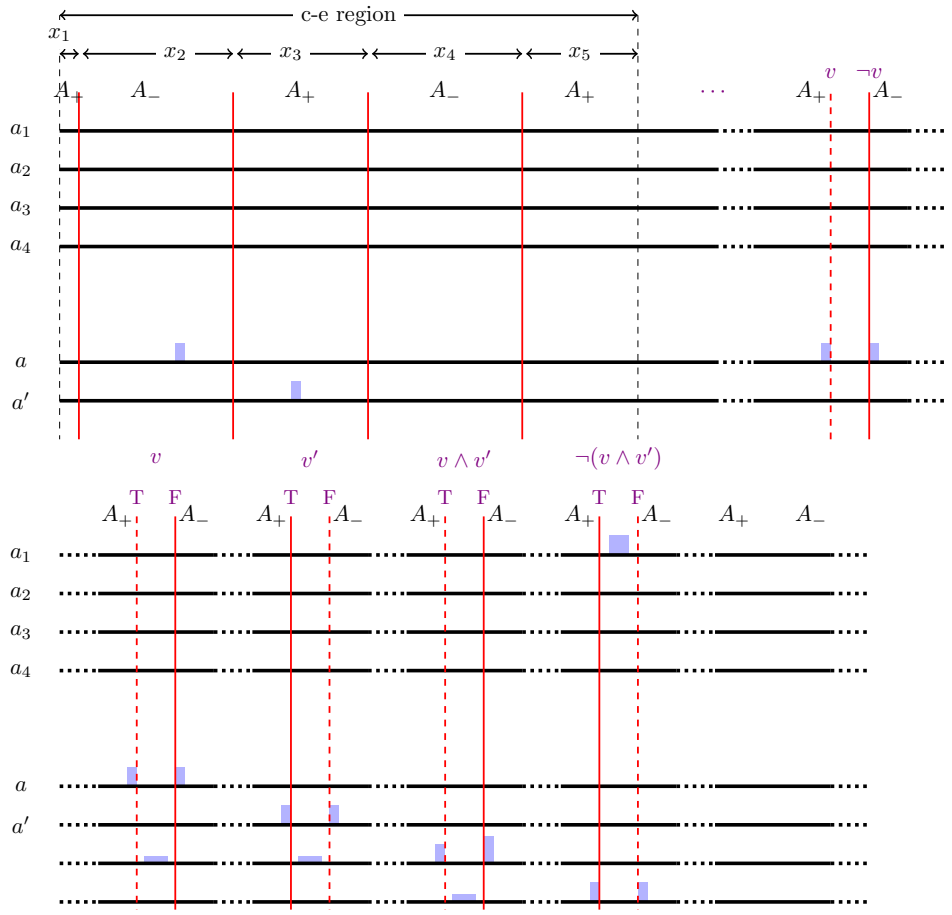


FIG. 15. *Example showing gate simulation: $n = 4$; agents $a$ and $a'$ have corresponding propositional variables $v$ and $v'$ that are two inputs to a circuit. $v =$* FALSE *since $a$'s sensor-value lies in a region labelled $A_-$; similarly $v' =$* TRUE *since $a'$'s sensor-value lies in a region labelled $A_+$. Gate-encoding blocks have cuts (shown in red) at two possible positions corresponding to* TRUE *and* FALSE*; a dashed-line shows the alternative position (*not *taken) by the cut (itself shown as a solid line). c-r agent $a_1$ receives feedback based on the conjunction of two input bits.*

*Comment.* Consider the operation of moving a cut from near the LHS of the c-e region to the RHS, which corresponds to two points in the Möbius-simplex that are close to each other via a path through the facets that have been identified according to Definition 4.12. Suppose also that within the c-e region, we do not change the label of any point. Then the tunnel-boundary sensor agents behave the same way: if some tunnel-boundary sensor agent sees an excess of $A_+$ in its interval, then it will continue to see an excess of $A_+$. Regarding the non-tunnel-boundary sensor agents, our reduction will make them "want" to produce opposite outputs, but due to the flipping of $x_{REF}$, the reference sensor value and the final output values produced by $g'_j$, $j \in \pm[n]$, are the same, and we will have continuity across this facet.

**5.1.4. Construction of circuit-encoders $C_2, \ldots, C_{p^C}$.** We next describe how the $p^C$ circuit-encoders differ from each other. Each $C_i$ has a set of circuit-encoding

agents $\mathcal{A}_i$, which contains $C_i$'s sensor agents $\mathcal{S}_i$. For $i \in [n]$ let $\mathcal{A}_i$ be the agents $a_{i,1}, \ldots, a_{i,p}$ for some polynomial $p$.

- For all $i, j$, $\mu_{a_{i,j}}(x) = \mu_{a_{1,j}}(y)$, where $x$ and $y$ are corresponding points in $R_i$ and $R_j$. By "corresponding points" here we mean points that lie in the same distance from the left-endpoint of the respective intervals $R_i$ and $R_j$.
  More concretely, we define a *correspondence function* $h_{\mathcal{R}_A,\mathcal{R}_B} : \mathcal{R}_A \longrightarrow \mathcal{R}_B$, mapping points of an interval $\mathcal{R}_A$ to an interval $\mathcal{R}_B$ in the most straightforward way: For $t \in \mathcal{R}_A$, let $h_{\mathcal{R}_A,\mathcal{R}_B}(t) = t - \ell(\mathcal{R}_A) + \ell(\mathcal{R}_B)$. In other words, any two points $x \in \mathcal{R}_A$ and $y \in \mathcal{R}_B$ such that $x - \ell(\mathcal{R}_A) = y - \ell(\mathcal{R}_B)$ are *corresponding points* with regard to the two subregions.
- For all $i, j$, all $x$ in the c-e region, $\mu_{a_{i,j}}(x)$, is specified in Definition 4.6.

The second of these items says that in the c-e region, the valuation function of the agents that make up $C_i$ differ from those of $C_1$ by having been shifted to the right by $\delta^{\text{tiny}}(i-1)$, where this shift wraps around in the event that we shift beyond $n$ (the right-hand point of the c-e region). In other respects, $C_i$ is an exact copy of $C_1$, save that $C_i$'s internal circuitry lies in $R_i$ rather than $R_1$.

For each $C_i$, the c-r agents have a further $2n$ value-blocks of value $1/(2p^C)$ in region $R_i$, whose labels are governed by the outputs produced by $C_i$ in the same way as for $C_1$. Consequently, we have the following observation.

*Observation* 5.2. The value that is labelled $A_+$ held by any c-r agent $a_j$ is the average of the output values that the $C_i$'s allocate to $a_j$. If, say, all the $C_i$ receive inputs representing a point in the Significant Region with label $\ell$, then $a_\ell$ observed an imbalance between $A_+$ and $A_-$, but $a_j$ for $j \neq \ell$ will have $g'_j$ output the opposite value to $g'_{-j}$, resulting in $a_j$'s value-blocks receiving opposite labels.

**5.2. An alternative coordinate system for the Möbius-simplex.** Recall that the Möbius-simplex $D$ is the $n$-simplex consisting of points $(x_1, \ldots, x_{n+1})$ whose components are nonnegative and sum to 1. Furthermore, a typical point in $D$ is directly encoded via the positions of $n$ cuts in the c-e region.

Here we specify a transformed coordinate system that is needed in order to encode instances of New Variant High-D Tucker. We will embed the hypercube-shaped domain of an instance of New Variant High-D Tucker in a hypercube in the transformed coordinates and then use properties of the transformed coordinate system to extend the labelling function to the rest of the domain in a way that does not introduce bogus solutions (i.e., fixpoints of the extended function that lie outside the hypercube and do not encode solutions of New Variant High-D Tucker).

Let $F_0$ be the set of points in $D$ of the form $(0, x_2, \ldots, x_n, 0)$; thus $F_0$ is a $(n-2)$-face of $D$. See Figure 16. For $\tau \in [0,1]$, let $\mathbf{x}_\tau$ be the point

$$\mathbf{x}_\tau := \tau(1, 0, \ldots, 0) + (1-\tau)(0, \ldots, 0, 1) = (\tau, 0, \ldots, 0, 1-\tau).$$

(So, $\mathbf{x}_0$ and $\mathbf{x}_1$ are the endpoints of the one-dimensional edge of $D$ that is not contained in $F_0$.) Let $D_\tau$ be the $(n-1)$-simplex consisting of convex combinations of $F_0$ and $\mathbf{x}_\tau$. Thus $D_0$ and $D_1$ are the two facets of $D$ that have been identified together as in Definition 4.12.

$D_\tau$ contains the point $\mathbf{0}_\tau = (\tau/n, 1/n, \ldots, 1/n, (1-\tau)/n)$, which we regard as the origin of $D_\tau$. The set of points $\{\mathbf{0}_\tau : 0 \leq \tau \leq 1\}$ will be referred to as the *axis*; it will transpire that all solutions must lie within an inverse polynomial distance from the axis (in particular, they will be in the Significant Region).

We then refer to points in $D_\tau$ by means of the coordinates in a coordinate system that itself is a linear function of $\tau$. With respect to any fixed $\tau \in [0,1]$ we define

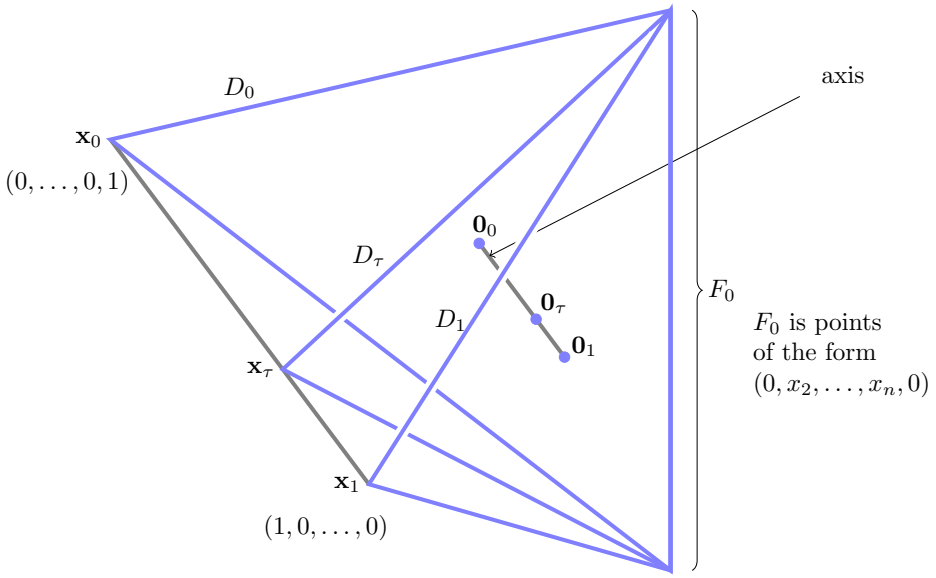


FIG. 16. *Subspaces of the Möbius-simplex $D$: $D_0$ is the triangle spanned by $\mathbf{x}_0$ and $F_0$, and $\mathbf{0}_0$ is its center, and similarly for $D_\tau$ and $D_1$.*

$n-1$ vectors $(d_2^\tau,\ldots,d_n^\tau)$ as follows. A key feature is that $(d_2^\tau,\ldots,d_n^\tau)$ form a basis of $D_\tau$ (so that with respect to the origin $\mathbf{0}_\tau$, any point in $D_\tau$ has unique coordinates). The other key feature (Observation 5.5) is that at $\tau=0$ the coordinate/directions are "equal and opposite" to the coordinates at $\tau=1$. Also, the coordinate system varies suitably smoothly.

As a warm-up we start by considering $d_2^\tau$:

$$d_2^\tau := (1-\tau)(0,1,-1,0,\ldots,0)+\tau(-1,1,0,\ldots,0).$$

$d_2^\tau$ consists of increasing the second coordinate at the expense of its neighbors. For small $\tau$ we increase mainly at the expense of the third coordinate, and as $\tau$ increases, we increase the second coordinate more at the expense of the first. Notice in particular that at $\tau=\frac{1}{2}$ we have $d_2^\tau=(-\frac{1}{2},1,-\frac{1}{2},0,\ldots,0)$.

Generally, for $2\le i\le n$ we define

$$d_i^\tau := (1-\tau)(\underbrace{0,\ldots,0}_{i-1\text{ zeroes}},1,-1,\underbrace{0,\ldots,0}_{n-i})+\tau(\underbrace{0,\ldots,0}_{i-2\text{ zeroes}},-1,1,\underbrace{0,\ldots,0}_{n-i+1}). \tag{5.1}$$

Thus, again this consists of the $i$th coordinate increasing at the expense of its neighbors, and we have in particular

$$d_i^{\frac{1}{2}}=\left(\underbrace{0,\ldots,0}_{i-2\text{ zeroes}},-\frac{1}{2},1,-\frac{1}{2},0,\ldots,0\right).$$

For $i=2,\ldots,n$, define

$$d_{-i}^\tau := -d_i^\tau. \tag{5.2}$$

Observation 5.3 makes the important point that by linearity, the vectors $d_i^\tau$, $i=2,\ldots,n$, can be used as a coordinate system to refer to points in $D_\tau$.

*Observation* 5.3. Any point $\mathbf{x}$ in $D_\tau$ can be uniquely expressed as a sum

$$\mathbf{x} = \mathbf{0}_\tau + \sum_{i=2}^{n} \alpha_i d_i^\tau. \tag{5.3}$$

To see this, note first that $D_0$ is points in $D$ of the form $(0, x_2, \ldots, x_{n+1})$, and $D_1$ is points of the form $(x_1, \ldots, x_n, 0)$. Note that the observation certainly works for $\tau = 0$ or $\tau = 1$. To see that it works for intermediate $\tau$, note that the vectors $d_i^\tau$ are linearly independent, and the reason why they span $D_\tau$ is that any vector $d_i^\tau$ is equal to $(1 - \tau)$ multiplied by a vector in $D_0$, added to $\tau$ multiplied by an equal-length vector in $D_1$. So these vectors do indeed lie in $D_\tau$.

DEFINITION 5.4. *For a point $\mathbf{x} \in D_\tau$ as in* (5.3) *we say that the* transformed coordinates *of $\mathbf{x}$ are $(\alpha_2, \ldots, \alpha_n)$. More generally, a point $\mathbf{x} \in D$ can be expressed as $(\tau; \alpha_2, \ldots, \alpha_n)$, where $\tau$ is chosen such that $\mathbf{x} \in D_\tau$. We use the following metric $\tilde{d}(\cdot, \cdot)$ on transformed coordinate vectors, where similarly to* (4.1)*, $L_1$ denotes the standard $L_1$ distance on vectors.*

$$\tilde{d}(\mathbf{x}, \mathbf{x}') = \min\Big(L_1(\mathbf{x}, \mathbf{x}'), \min_{\mathbf{z}, \mathbf{z}' : \mathbf{z} \equiv \mathbf{z}'} (L_1(\mathbf{x}, \mathbf{z}) + L_1(\mathbf{z}', \mathbf{x}'))\Big), \tag{5.4}$$

*where* $(0; \alpha_2, \ldots, \alpha_n) \equiv (1; -\alpha_2, \ldots, -\alpha_n)$.

*Observation* 5.5. With regard to Definition 5.4, let us consider two points $\mathbf{x} = (0; \alpha_2, \ldots, \alpha_n)$ and $\mathbf{x}' = (1; -\alpha_2, \ldots, -\alpha_n)$ that have been equated with each other.

Assume these points are near the axis, specifically $|\alpha_j| < 1/10n$ for all $j$. Notice that

- the cuts in the c-e region for $\mathbf{x}$ and $\mathbf{x}'$ partition the c-e region in the same way;
- (with reference to Figure 17) when we move from a point in $D_{1-\varepsilon}$ to a nearby point in $D_\varepsilon$, for any $j \in \{2, \ldots, n\}$, the direction of increasing $\alpha_j$ segues *smoothly* to the direction of *decreasing* $\alpha_j$.

Our assumption that $|\alpha_j| < 1/10n$ ensures that cuts are fairly evenly spaced, and movement in any of the directions $d_j^\tau$ does not cause the cuts to cross each other.

Proposition 5.6 says that if we perturb a point $\mathbf{x} \in D$ that lies close to the axis, then the total perturbation of the transformed coordinates of $\mathbf{x}$ is polynomially related to the total perturbation of the untransformed coordinates; $d$ and $\tilde{d}$ are polynomially related.

PROPOSITION 5.6 (polynomial distance relation). *There is some polynomial $p(n)$ such that for all $\mathbf{x}, \mathbf{x}' \in D$ within Euclidean distance $1/10n^2$ of the axis, letting $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{x}}'$ be their transformed coordinates, and letting $d, \tilde{d}$ be the metrics defined as in (Equations* (4.1)*,* (5.4)*), we have*

$$\frac{1}{p(n)} \leq \frac{d(\mathbf{x}, \mathbf{x}')}{\tilde{d}(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}')} \leq p(n).$$

*Proof.* In subsection 5.2.1, we show that for points near the axis (i.e., within Euclidean distance $1/10n^2$ of the axis), the computation of the coordinate transformation—and its inverse—have the property that small perturbations of the input values lead to inverse-polynomial upper bounds on the resulting perturbations of the output values. Since we have this for both the transformation and its inverse,

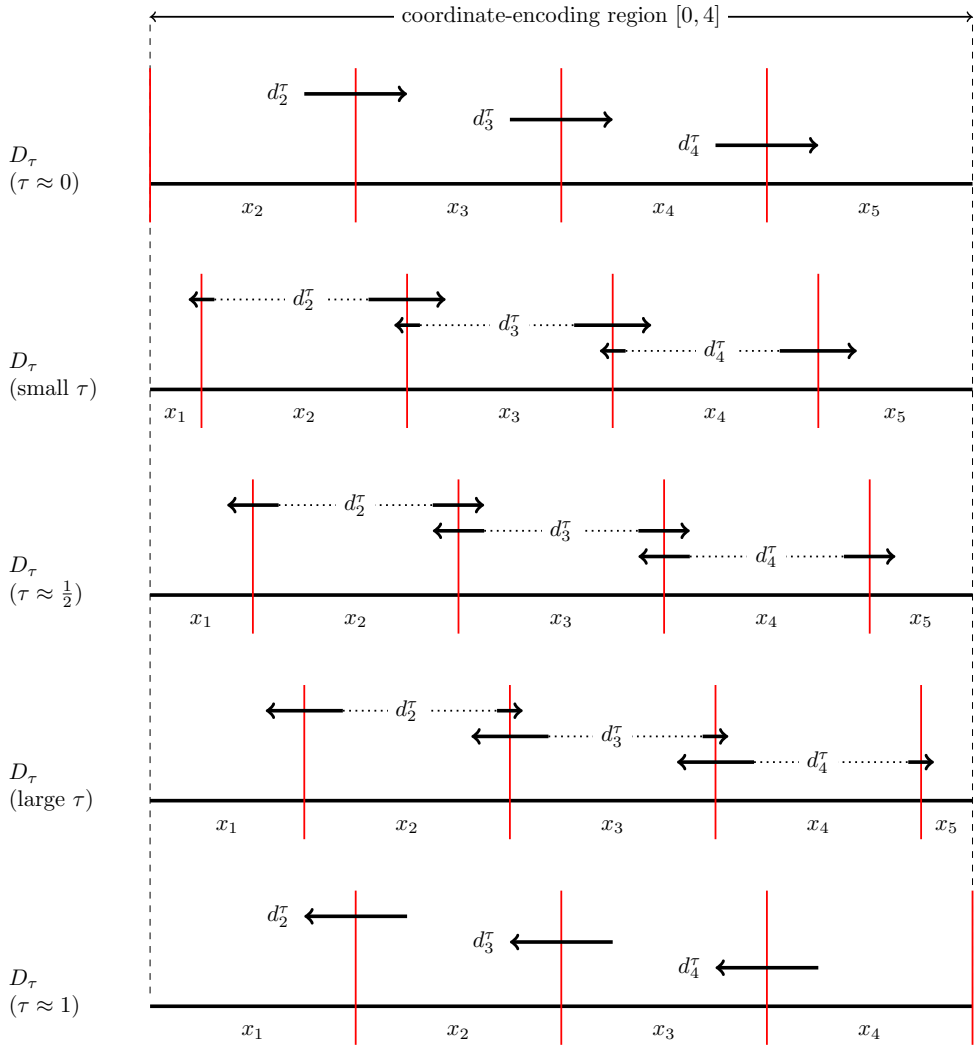


FIG. 17. *The diagram shows (for $n = 4$) sets of cuts (in red) that correspond to points on the axis, for various values of $\tau$. It also shows how movements of the cuts correspond to movement of a point in $D$ away from the axis. For example, for small $\tau$, a move in direction $d_2^\tau$ corresponds to moving the second cut to the right and the first only slightly to the left. Generally, a movement in direction $d_i^\tau$ tends to increase $x_i$ at the expense of $x_i$'s neighbors $x_{i-1}$ and $x_{i+1}$. As $\tau$ increases, the movement in direction $d_i^\tau$ tends increasingly to moving the cut to the left of interval $x_i$ to the left, as opposed to moving the cut to the right of interval $x_i$ to the right. In the limit as $\tau$ approaches 0 from above, the direction $d_i^\tau$ approaches the* negative *of the limit approached by $d_i^\tau$ when $\tau$ approaches* 1 *from below.*

it follows that there are also inverse-polynomial lower bounds on the resulting perturbations of the output values.

The identification of transformed coordinates $(0; \alpha_2, \ldots, \alpha_n)$, $(1; -\alpha_2, \ldots, -\alpha_n)$ is of course equivalent to the identification of untransformed coordinates $(0, x_1, \ldots, x_n)$ and $(x_1, \ldots, x_n, 0)$ in (4.1). If, say, $\mathbf{x}$ and $\mathbf{x}'$ are very close together due to being linked via $\mathbf{z}, \mathbf{z}'$ for which $\mathbf{z} \equiv \mathbf{z}'$, then the transformed versions $\tilde{\mathbf{z}}, \tilde{\mathbf{z}}'$ would cause $\tilde{d}(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}')$ to be very close. So the polynomially related result for this space in which these two

facets have not been identified with each other carries over to a polynomially related result in which they have been identified with each other. ∎

Note that a similar result would hold if in the definitions of the metrics $d$ and $\tilde{d}$, we replace the $L_1$ metric with, say, $L_2$ or $L_\infty$, since these are polynomially related to $L_1$. Note that Proposition 5.6 does not hold for all points in $D$; the restriction to a neighborhood of the axis is needed. For points on $F_0$, all values of $\tau$ are equivalent, and for points close to $F_0$, perturbed versions of them could result in large perturbations of $\tau$.

We show in the next section that the coordinate transformation, and its inverse, can be computed in polynomial time, for points in $D$ that are within some inverse polynomial distance from the axis.

**5.2.1. Computation of the transformation and its inverse.** We verify here that (for points in the vicinity of the axis), our transformation may be performed efficiently and that small perturbations of inputs lead to small perturbations of the outputs (in either direction). The easy direction is the computation of $(x_1, \ldots, x_{n+1})$ from $(\tau; \alpha_2, \ldots, \alpha_n)$. Recall that a point $\mathbf{x}$ on the original domain can be expressed in terms of the transformed coordinates $(\tau; \alpha_2, \ldots, \alpha_n)$ and the origin $\mathbf{0}_\tau = (\frac{\tau}{n}, \frac{1}{n}, \ldots, \frac{1}{n}, \frac{1-\tau}{n})$ as $\mathbf{x} = \mathbf{0}_\tau + \sum_{i=2}^{n} \alpha_i d_i^\tau$. Therefore, we have

$$
\begin{aligned}
x_1 &= \frac{\tau}{n} - \tau.\alpha_2, \\
x_2 &= \frac{1+\tau}{n} + (1-\tau)\alpha_2 - \tau\alpha_3 - x_1, \\
x_3 &= \frac{2+\tau}{n} + (1-\tau)\alpha_3 - \tau\alpha_4 - (x_1 + x_2), \\
&\vdots \\
x_n &= \frac{n-1+\tau}{n} + (1-\tau)\alpha_n - \sum_{i=1}^{n-1} x_i, \\
x_{n+1} &= 1 - \sum_{i=1}^{n} x_i.
\end{aligned}
$$

In the other direction, given $(x_1, \ldots, x_{n+1})$ we first compute the value of $\tau$ for the transformed coordinate system: Note that $(x_1, \ldots, x_{n+1})$ must be a convex combination of $x_\tau$ and $F_0$ (where recall that $x_\tau = (\tau, 0, \ldots, 0, 1-\tau)$ and $F_0$ is points of the form $(0, x_2, \ldots, x_n, 0)$); therefore, $\tau$ can be computed as the solution to the equation

$$\frac{\tau}{1-\tau} = \frac{x_1}{x_{n+1}}.$$

Note that the dependence of $\tau$ on $x_1$ and $x_{n+1}$ is not excessively sensitive near the axis, since $x_1 + x_{n+1}$ is close to $1/n$. Having computed $\tau \in [0, 1]$, we could simply solve the equations above (the ones used to compute $x_1, \ldots, x_{n+1}$) for $\alpha_2$, $\alpha_3$, and so on successively, using the derived formulas for $\alpha_i$ for the computation of $\alpha_{i+1}$, and express each $\alpha_i$ as a function of only $\tau$ and the values $x_1, x_2, \ldots x_{n+1}$. However, in the extremal cases of $\tau = 0$ and $\tau = 1$, some of the $\alpha_i$ values might "disappear"; for example, for $\tau = 0$, expressing $\alpha_3$ in terms of only $\tau$, $x_1$, and $x_2$ is not possible, since for $\tau = 0$ we do not obtain a formula for $\alpha_2$ to substitute into the equation for $x_2$. To remedy this, we consider two cases:

*Case* 1: $\tau \geq \frac{1}{2}$. We compute $\alpha_2, \ldots, \alpha_n$ as follows:

$$\begin{aligned}\alpha_2 &= \frac{1}{n} - \frac{1}{\tau} \cdot x_1,\\ \alpha_3 &= \frac{1 + 1/\tau}{n} + \frac{(1-\tau)}{\tau} \cdot \alpha_2 - \frac{1}{\tau} \cdot (x_1 + x_2),\\ &\vdots\end{aligned}$$

and so on for $\alpha_4, \ldots, \alpha_n$.

*Case* 2: $\tau \leq \frac{1}{2}$. We compute $\alpha_2, \ldots, \alpha_n$ starting at the opposite end:

$$\begin{aligned}\alpha_n &= -\frac{n-1+\tau}{n(1-\tau)} + \frac{\sum_{i=1}^{n} x_i}{1-\tau},\\ \alpha_{n-1} &= \frac{\tau}{1-\tau} \cdot \alpha_n - \frac{n-2+\tau}{n(1-\tau)} + \frac{\sum_{i=1}^{n-1} x_i}{1-\tau},\\ &\vdots\end{aligned}$$

and so on for $\alpha_{n-2}, \ldots, \alpha_2$.

Note that inverse polynomial-size perturbations of the $x_i$ lead to inverse perturbations of the transformed coordinates of polynomial size. As a sanity check, note that at the boundary (points with $\tau = 0$ are the same as points with $\tau = 1$), if we move a cut at the LHS to the RHS (so $(0, x_2, \ldots, x_{n+1})$ becomes $(x_2, \ldots, x_{n+1}, 0)$), it can be checked that the $\alpha_i$ get negated.

Note that these computations should be done with a precision (or rounding error) polynomially smaller than $\delta^{\text{tiny}}$.

**5.3. A (poly-time computable) partial colouring function.** This section defines a partial function $f : D \to \{-1, 0, 1\}^n$ ($D$ being the $n$-dimensional Möbius-simplex (Definition 4.12)). $f$ is constructed in polynomial time based on an instance $I_{VT}$ of New Variant High-D Tucker in $n$ dimensions, defined using circuit $C_{VT}$. $f$ is defined in the Significant Region (Definition 4.14), which is the set of points where no tunnel-boundary sensor agents (Definition 4.6) are active, and thus it includes the twisted tunnel $T$ (Definition 5.7). $f$ is computable by a circuit $C$, which is used to define the operations of the circuit-encoders in a derived instance of Consensus-Halving. Within the Significant Region, $f$ determines the outputs of the circuit-encoders.

The function $f$ maps a point $\mathbf{x}$ in the Significant Region to a vector of length $n$, $e_f(\mathbf{x})$, where $e_f(\mathbf{x})_j = 1$ means that the point receives color $j$, $e_f(\mathbf{x})_j = -1$ means that the point receives color $-j$, and $e_f(\mathbf{x})_j = 0$ means that the point does not receive color $j$ or $-j$. In general, $e_f(\mathbf{x})$ may have multiple nonzero entries. We will use the term *the color of* $\mathbf{x}$ for points that only receive a single color (and therefore their outputs are vectors with only one nonzero entry).

The function $f$ will have a corresponding vector-valued function $f'$ (subsection 6.1) that more closely represents choices of labels $A_+/A_-$ that the circuit shows to the c-r agents. We will do this in such a way that no bogus solutions result from the transition to parts of $D$ where tunnel-boundary sensor agents are active. By construction, there are no solutions where tunnel-boundary sensor agents are active, so all solutions occur where $f$ is defined.

In subsection 6.1 we then define a vector-valued "Borsuk–Ulam-style" function $F$ in terms of $f$. Letting $I_{VT}$ be an instance of New Variant High-D Tucker, $F(\mathbf{x})$ will be approximately zero iff given $\mathbf{x}$, we can derive an approximate consensus-halving solution to $I_{VT}$. It will be shown that approximate zeroes of $F$ provide solutions to $I_{VT}$.

Recall that $D$ is the set of points $(x_1, \ldots, x_{n+1})$ whose components are nonnegative and sum to 1. And, the Significant Region (Definition 4.14) of $D$ consists of points $(x_1, \ldots, x_{n+1})$ for which coordinates $x_i$ $(2 \leq i \leq n-1)$ differ from $1/n$ by at most an inverse polynomial $\delta^w = 1/p^w(n)$ (Proposition 4.15). ($\delta^w$ represents an upper bound on the thickness of the Significant Region.)

Let $B$ be the $n$-dimensional "box" associated with $I_{VT}$ (recall $I_{VT}$ is represented by circuit $C_{VT}$ that maps points in $B$ to $\pm[n]$). We embed a copy of $B$ in $D$ as follows. Recall the way facets of $B$ are colored in Definition 3.5. Let $(x_1, \ldots, x_n)$ denote a typical point in $B$, and assume that the facets of $B$ with maximum and minimum $x_1$ (i.e., $x_1 = 1$ and $x_1 = -1$, respectively) are the panchromatic facets of $B$ (as in Definition 3.5), and for $i \geq 2$ the facet of $B$ with maximum $x_i$ ($x_i = 1$) consists of points that do not have color $i$, and the facet of $B$ with minimum $x_i$ ($x_i = -1$) consists of points that do not have color $-i$.

Definition 5.7. *The* twisted tunnel $T$ *is defined as follows. The* axis *of* $T$ *is the set of all points* $\mathbf{0}_\tau$ *as defined in subsection* 5.2. *The twisted tunnel is the set of all points with transformed coordinates* $(\tau; \alpha_2, \ldots, \alpha_n)$ *such that for all* $i$, $|\alpha_i| < \delta^T$. *Note that* $\delta^T$ *is an inverse polynomial quantity sufficiently small that* $T$ *is a subset of the Significant Region; this is achieved since by definition,* $\delta^T$ *is polynomially smaller than* $\delta^w$ *of Proposition* 4.15. *Thus,* $T$ *has (with respect to the transformed coordinates) a* $(n-1)$*-cube-shaped intersection with any* $D_\tau$.

We define the behavior of $f$ over the Significant Region (Definition 4.14) in three stages, as follows.

1. *Embedding* $B$ *in* $D$ *(recall* $B = [-1, 1]^n$*).*
   A point $\mathbf{x} = (x_1, \ldots, x_n)$ in $B$ is mapped to a point $g(\mathbf{x})$ in $D$ as follows. $g(\mathbf{x})$ lies in $D_\tau$, where we choose $\tau = \frac{1}{2} + \delta^T \cdot x_1$. Then (noting (5.3)) we set $g(\mathbf{x})$ equal to $\mathbf{0}_\tau + \sum_{i=2}^n \delta^T \cdot x_i d_i^\tau$ (i.e., $g(\mathbf{x})$ has transformed coordinates $(\frac{1}{2}+\delta^T x_1; \delta^T x_2, \ldots, \delta^T x_n)$). $g(\mathbf{x})$ will receive a single color; the color of $g(\mathbf{x})$—i.e., the nonzero entry of $f(g(\mathbf{x}))$—is set equal to the color allocated to $\mathbf{x}$ in $B$ by $I_{VT}$. (Notice that the center of $B$ is mapped to $(1/2n, 1/n, \ldots, 1/n, 1/2n)$, which is the origin of $D_{\frac{1}{2}}$, and the center of the Significant Region. This point has (recalling Definition 5.4) transformed coordinates $(\frac{1}{2}; 0, \ldots, 0)$, where the first entry is the value of $\tau$.)
2. *Extending* $f$ *to be defined on* $T$.
   We also color other points in $T$ as follows—these will also receive single colors. Suppose $\mathbf{y}$ belongs to $D_\tau$, where $\tau < \frac{1}{2} - \delta^T$ or $\tau > \frac{1}{2} + \delta^T$. According to (5.3), $\mathbf{y} = \mathbf{0}_\tau + \sum_{i=2}^n \alpha_i d_i^\tau$, and $\mathbf{y}$ has transformed coordinates $(\tau; \alpha_2, \ldots, \alpha_n)$. Suppose all the $\alpha_i$ lie in the range $[-\delta^T, \delta^T]$. Then if $\tau < \frac{1}{2} - \delta^T$, we set the color of $\mathbf{y}$ to the color of a point $\mathbf{y}' = (\frac{1}{2} - \delta^T; \alpha_2, \ldots, \alpha_n)$. Thus $\mathbf{y}' \in D_{\frac{1}{2}-\delta^T}$, and the other transformed coordinates (Definition 5.4) are the same for $\mathbf{y}$ and for $\mathbf{y}'$. We do a similar thing for points in $D_\tau$ for $\tau > \frac{1}{2} + \delta^T$. That is, if $\mathbf{y}$ has transformed coordinates $(\tau; \alpha_2, \ldots, \alpha_n)$, where $\tau > \frac{1}{2} + \delta^T$ and the $\alpha_i$ are all at most $\delta^T$ in absolute value, then $\mathbf{y}$ gets the same color as a point $\mathbf{y}'$ whose transformed coordinates are $(\frac{1}{2} + \delta^T; \alpha_2, \ldots, \alpha_n)$.

3. *Extending $f$ to the Significant Region.*
   The Significant Region (Definition 4.14) is points in $D$ where no tunnel-boundary sensor agents are active, a subset of points that are close to the axis in the sense of Proposition 4.15. Consider $\mathbf{x} \in D \setminus T$ with transformed coordinates $(\tau; \alpha_2, \ldots, \alpha_n)$.
   (a) For each $j \in \{2, \ldots, n\}$ if $\alpha_j > \delta^T$, then $\mathbf{x}$ gets color $-j$.
   (b) For each $j \in \{2, \ldots, n\}$ if $\alpha_j < -\delta^T$, then $\mathbf{x}$ gets color $j$.
   (c) These are not mutually exclusive; $\mathbf{x}$ gets at least one color, possibly more.
   Notice that (within the subspace $D_\tau$) the side(s) of the twisted tunnel $T$ closest to $\mathbf{x}$ is guaranteed not to be opposite to any color of $\mathbf{x}$.
   For a subset $S$ of colors, let $R(S)$ be the region with colors in $S$. We call these the "outer regions."

Proposition 6.11 notes that when color-regions meet each other at opposite ends of $T$ (which have been identified with each other according to the definition of the Significant Region), they will have equal and opposite colours.

**5.4. How to compute a solution to** New Variant High-D Tucker **from a solution to** Consensus-Halving**.** Suppose we have a solution $S_{CH}$ to an instance $I_{CH}$ of Consensus-Halving, derived by our reduction from an instance $I_{VT}$ of New Variant High-D Tucker.

Let $\mathbf{x}$ be the point in the Möbius-simplex represented by the c-e cuts of $S_{CH}$. Proposition 4.15 already tells us that $\mathbf{x}$ must lie within some inverse-polynomial distance of the axis, since if not, some tunnel-boundary sensor agent will be active. We prove in section 6 that $\mathbf{x}$ must lie within, or very close to, the twisted tunnel.

From this we identify two color-regions that have equal and opposite colours, as follows. Let $\mathbf{x}$ have transformed coordinates $(\tau; \alpha_2, \ldots, \alpha_n)$, which can be computed with inverse-polynomial precision from the c-e cuts.

If $\mathbf{x}$ occurs within the embedded copy of $B$ (subsection 5.3), then identify two circuit-encoders $C_i$ and $C_{i'}$ that both receive reliable inputs and have equal and opposite outputs. The proof of Proposition 6.7 tells us that this is always possible.

Now we have two points $\mathbf{x}'$ and $\mathbf{x}''$ within distance $\delta^{\text{tiny}}$ of each other that lie in oppositely colored cubelets. With respect to transformed coordinates, $\mathbf{x}$ and $\mathbf{x}'$ are within some distance $\tilde{\delta}^{\text{tiny}}$ that we can assume by Proposition 5.6 to be much smaller than the widths of the cubelets and other color-regions. So these two cubelets are adjacent and we are done.

If $\mathbf{x}$ lies in $T$ but not in the embedded copy of $B$, then we similarly find two distinct color-regions that are adjacent and with opposite colors. Since these color-regions are just extensions of the cubelets that lie on the panchromatic facets of the embedded instance of New Variant High-D Tucker, we are done. Formally, if $S_{CH}$ represents a point with transformed coordinates $(\tau; \alpha_2, \ldots, \alpha_n)$, where $\tau > \frac{1}{2} + \delta^T$, we take the point $(\frac{1}{2} + \delta^T; \alpha_2, \ldots, \alpha_n)$, and similarly, if $S_{CH}$ represents a point with transformed coordinates $(\tau; \alpha_2, \ldots, \alpha_n)$, where $\tau < \frac{1}{2} - \delta^T$, we take the point $(\frac{1}{2} - \delta^T; \alpha_2, \ldots, \alpha_n)$.

**6. Establishing the correctness of the reduction.** Let $I_{VT}$ be an instance of New Variant High-D Tucker in $n$ dimensions, given in terms of circuit $C_{VT}$; suppose Consensus-Halving instance $I_{CH}$ is derived from it by our reduction. Recall that $\varepsilon = \delta^{\text{tiny}}/10$. We show that any $\varepsilon$-approximate solution $S_{CH}$ to $I_{CH}$ allows a solution to $I_{VT}$ to be recovered using subsection 5.4.

In $S_{CH}$, only the c-e cuts may lie in the c-e region (Observation 4.9), and every other cut $c(a)$ for $a$ not a c-r agent must lie in some interval outside the c-e region (in order for $a$'s value to be evenly split). It follows from Proposition 4.15 that at least $n-1$ c-e cuts must lie in the c-e region, and they are evenly spaced (the gaps between them differ from 1 by an inverse polynomial). The remaining cut may occur elsewhere, and in that case, we refer to it as a "stray cut."

DEFINITION 6.1 (stray cut). *In a solution $\mathcal{H}$ of $I_{CH}$, a c-e cut will be called a* stray cut *if it occurs outside of the c-e region.*

A stray cut may have two effects on $\mathcal{H}$.

1. It intersects the circuit-encoding region $R_i$ of some circuit encoder $C_i$ for $i \in \{1, \ldots, p^C\}$.
2. It flips the parity of the circuit encoders $C_i$ with $R_i < c$, where $c$ is the position of the stray cut in $R_{i-1}$. In other words, if the first cut in $R_i$ was expecting to see $A_+$ on its LHS, it now sees $A_-$ and vice versa.

The first effect is not much of a problem; we simply deem this circuit unreliable.

DEFINITION 6.2 (reliable copy). *We will say that a copy $C_i$ of the circuit $C$ ($i \in \{1, \ldots, p^C\}$) is* reliable *if none of the c-e cuts intersect $R_i$. A copy $C_i$ of the circuit is* unreliable *if it is not reliable.*

Since there is only one stray cut, there is at most one unreliable circuit $C_i$. The error that this copy will introduce to the volumes of the labels $A_+$ and $A_-$ for the c-r agents (see subsection 5.1.2) will be relatively small due to the fact that there are many reliable points that receive good inputs. This follows from the presence of $p^C$ copies of the circuitry and the averaging argument over the outputs of these circuits; see subsection 5.1.4 and Observation 4.13. Effectively, since every circuit-encoder reads an input which represents a point in the Möbius-simplex (see Observation 4.13) and outputs the label of the point, an unreliable copy can cause one point to be mislabelled. However, since the $p^C$ points are close enough, this is not enough to prevent a solution, as there will be other points close by that will be correctly labelled, since their corresponding copies are reliable.

The second effect from the ones above is potentially more troublesome, since the parity flip could alter the outputs of the sensor agents. This problem, however, is actually being taken care of by the reference sensor agent and the value $x_{REF}$. If the outputs of the sensor agents are flipped, the circuit actually inputs the bitwise complements of what it would input before the flip; these consist of binary strings that encode the positions of the cuts. The effects of these flips cancel out and the circuit outputs exactly the same label, which is then flipped by the XOR subcircuit to ensure that the c-r agents receive the same feedback. We have the following lemma.

LEMMA 6.3 (double-negative lemma). *Consider a solution $\mathcal{H}$ of $I_{CH}$ and a circuit-encoder $C_i$. If a stray cut is placed in $(1, \ell(R_i))$ (i.e., to the right of the c-e region and to the left of $R_i$), then the c-r agents see exactly the same balance of $A_+$ and $A_-$ in $R_i$ as they did before the insertion of the stray cut.*

*Proof.* Consider the operation of adding a cut between the c-e region and $R_i$. This effectively causes the output bits of the sensor agents of $C_i$ to flip, as the cut in the output interval for every sensor agent $s_{i,j} \in S_i$ is now "seeing" $A_+$ on its LHS, rather than $A_-$. In other words, the sensor agents now output the complement of the bit value that they would have produced, if they were seeing $A_+$ on their LHS.

In particular, the first sensor agent $s_{1,j}$ now sees $A_-$ on its LHS, rather than $A_+$, and produces the complement of its original value. Note, however, that this agent is actually the reference sensor agent and recall that $x_{REF} \in \{\text{TRUE}, \text{FALSE}\}$ is the complement of the value that it produces; in this case, this its original value, before the flip.

If we were to use the output of $C_i^{VT}$ directly to provide feedback to the c-r agents, then the following undesired effect would take place. Comparing the situations before and after the stray cut, the balance of $A_+$ and $A_-$ shown to the c-r agents in $R_i$ would flip, because (i) the output of $C_i^{VT}$ is unaffected by the flip but (ii) the stray cut changes the parity of the label sequence, causing the parts of $R_i$ (which are value blocks of the c-r agents) that were labelled $A_+$ to now be labelled $A_-$ and vice versa. That would introduce a false discrepancy of the balance of $A_+$ and $A_-$ for one of the c-r agents, or more precisely, the correct "amount" of discrepancy but in the wrong direction.

However, as explained in subsection 5.1.2, the output gates $g'_j$ are designed via the appropriate combination of the actual value of the gate $g_j$ and $x_{REF}$. In particular, let $g_j$ be the gate that is set to TRUE by the circuit. For any other gate $g_h$, we will have $g'_{|h|} = g'_{-|h|}$ as before, so the balance of $A_+$ and $A_-$ that the c-r agents see are the same as before. If $j > 0$, then $g'_j$ and $g'_{-j}$ will be set to $\text{TRUE} \oplus x_{REF}$ and they will have the opposite value compared to what they had before the flip. For example, if $x_{REF}$ was TRUE before the flip, then $g_j$ and $g'_j$ would be FALSE before the flip, forcing the corresponding cuts to lie on the RHS of the value blocks of the corresponding c-r agent. After the flip, since $x_{REF}$ has the opposite value (e.g., FALSE), $g_j$ and $g'_j$ will assume the opposite value (e.g., TRUE), forcing the corresponding cuts to lie on the oppose side (e.g., the LHS) of the value blocks of the corresponding agent. However, since a flip on the parity of labels has also taken place, the corresponding c-r agent receives exactly the same feedback as before.

This is the "double-negative" effect of the lemma. ∎

Recalling Observation 4.13, the c-e cuts of $S_{CH}$ encode a collection of $p^C$ points $\mathbf{x}_1, \ldots, \mathbf{x}_{p^C}$ in the Möbius-simplex, where $d(\mathbf{x}_i, \mathbf{x}_j) \leq \delta^{\text{tiny}}$ ($d$ is the metric defined in (4.1)). In transformed coordinates we have $\tilde{d}(\mathbf{x}_i, \mathbf{x}_j) \leq \tilde{\delta}^{\text{tiny}}$ ($\tilde{d}$ as in (5.4)), and by Proposition 5.6, $\tilde{\delta}^{\text{tiny}}$ is much smaller than other inverse-polynomial quantities that we work with. Recall also that at most $n$ of these points are incorrectly labelled since all but $n$ of the circuit-encoders receive reliable inputs. Alternatively (if there is a stray cut), up to $n-1$ circuit-encoders receive unreliable input and one circuit-encoder is affected by the stray cut. We proceed by case analysis with respect to the location of the points $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Note that there is an inverse-polynomial gap between the twisted tunnel and the boundary of the Significant Region, which is much larger than $\delta^{\text{tiny}}$. So the cases to consider are as follows:

- The points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ lie in the Significant Region. In this case, Proposition 6.7 shows that some of them must lie in the twisted tunnel, and the procedure of subsection 5.4 identifies a solution to NEW VARIANT HIGH-D TUCKER; see subsections 6.1 and 6.2.
- Some but not all of the points are outside the Significant Region. In that case we are far from the twisted tunnel, and subsection 6.3 argues that no solution is possible here.
- All points are outside the Significant Region. In this case a solution can be straightforwardly ruled out, since various tunnel-boundary sensor agents

will be active, and the points are so close together (within $\delta^{\text{tiny}}$) that the directions in which they are active cannot cancel.

**6.1. A Borsuk–Ulam-style function $F : D \to [-1,1]^n$.** Recall that $D$ denotes the $n$-dimensional Möbius-simplex (Definition 4.12). We start by defining a function $f' : D \to [-1,1]^n$ based on $f$ defined as in subsection 5.3. $f'$ simulates the effect of the tunnel-boundary sensor agents as described in subsection 5.1.3. Let $\mathbf{x} \in D$.

1. *When no tunnel-boundary sensor agents are active at* $\mathbf{x}$. Here we are in the Significant Region.
   - In $T$, $f'$ behaves like $f$ in the sense that if $f$ assigns the color $i$ to $\mathbf{x}$ (recall that it assigns a single color to points in $T$), then set $f'(\mathbf{x}) := \mathbf{e}_i$ if $i > 0$, $f'(\mathbf{x}) := -\mathbf{e}_{|i|}$ for $i < 0$.
   - Outside $T$, in outer region $R(S)$, $f'(\mathbf{x}) := \sum_{i \in S, i>0} \mathbf{e}_i + \sum_{i \in S, i<0} -\mathbf{e}_{|i|}$.
2. *When one or more tunnel-boundary sensor agents are active.* If the $j$th tunnel-boundary sensor agent of $C_1$, $b_{1,j}$ is active toward $A_+$ (respectively, $A_-$), then the $j$th entry of $f'(\mathbf{x})$ is set to 1 if $j$ is odd and to $-1$ if $j$ is even (respectively, to $-1$ if $j$ is odd and 1 is $j$ is even). This is done for all active tunnel-boundary sensor agents, and thus $f'(\mathbf{x})$ can contain multiple 1's and $-1$'s.

The following points are similar to Observation 4.13.

*Observation* 6.4. Suppose that circuit-encoder $C_i$ (some $i \in [p^C]$) of $I_{CH}$ receives reliable inputs. (Observation 4.11 tells us that at most $n$ of them fail to receive reliable inputs.) Then $C_i$ computes $f'$ at a point within distance $\delta^{\text{tiny}}$ from the $\mathbf{x} \in D$ encoded by the c-e cuts, in the sense that the value observed by each c-r agent $a_j$ that is labelled by $A_+$, minus the amount labelled $A_-$, restricted to that part of $a_j$'s value that lies in $R_i$ and so is governed by the output of $C_i$, is the $j$th component of $f'$.

This follows from the construction of subsection 5.1.2 and the association of boolean values TRUE, FALSE with the labels $A_+$ and $A_-$. For $\mathbf{x} \in D$, $F(\mathbf{x})$ is the average of the outputs of the $C_i$; Proposition 6.5 provides the details.

PROPOSITION 6.5. *$I_{CH}$ computes a function $F$ in the following sense. Let $\mathbf{x}$ be the point encoded by the c-r agents. Suppose all agents other than the c-r agents have error (i.e., discrepancy between $A_+$ and $A_-$ that they observe) at most $\varepsilon$. Then the error of the c-r agents is within additive distance $1/n^2$ from the average value of $f'$, averaged over a set of points all within $\delta^{\text{tiny}}$ of $\mathbf{x}$.*

*Proof.* We put together various observations about the way $I_{CH}$ is constructed. Observation 4.13 told us that the values observed by the c-r agents are the average of a set of points all within distance $\delta^{\text{tiny}}$ of each other. The additive distance $1/n$ results from the existence of up to $n$ circuit-encoders that either fail to receive good inputs (Observation 4.11) or are affected by the stray cut, taken in conjunction with the fact that we average over $p^C$ points, where $p^C$ can be taken to be at least $2n^3$. ∎

PROPOSITION 6.6. *$\delta^{\text{tiny}}$ can be chosen to be sufficiently small (but still inverse-polynomial) such that given a set of $p^C$ points $\mathbf{x}_1, \ldots, \mathbf{x}_{p^C} \in D$ within distance $\delta^{\text{tiny}}$ of each other, when we compute their transformed coordinates $\mathbf{y}_1, \ldots, \mathbf{y}_{p^C}$, we have the following:*

> *Every pair of points $\mathbf{y}, \mathbf{y}' \in \{\mathbf{y}_1, \ldots, \mathbf{y}_{p^C}\}$ has the property that they either lie in the same color-region, adjacent color-regions (where a "color-region" is one of the monochromatic regions of subsection 5.3), or one of the outer regions.*

*Proof.* Recall the coloring of the Significant Region as defined in subsection 5.3, and recall the color-regions are part of the Significant Region. This means that any point that lies in these regions should be within an inverse-polynomial quantity $\delta^w$ of the axis. Since $\delta^w$ can be chosen to be smaller than $1/10n^2$, Proposition 5.6 establishes that any two points $\mathbf{y}_i$ and $\mathbf{y}_j$ are within distance at most $p(n) \cdot d(\mathbf{x}_i, \mathbf{x}_j) \leq p(n) \cdot \delta^{\text{tiny}}$, where $\mathbf{x}_i$ and $\mathbf{x}_j$ are the points of which $\mathbf{y}_i$ and $\mathbf{y}_j$ are the points in transformed coordinates and $p(n)$ is the polynomial of Proposition 5.6. At the same time, the distance between any two points in two different and nonadjacent color-regions (the "thickness" of these regions) is lower bounded by some inverse-polynomial quantity $1/q(n)$, by the way these regions were defined (based on the coloring function) in subsection 5.3. Essentially, there are various value thresholds (on the coordinate values of a point $\mathbf{y}$ in transformed coordinates) which dictate which color-region a point belongs to, and these threshold values differ from each other by inverse-polynomial amounts, with the smallest difference between any of them being inverse-polynomial. $\delta^{\text{tiny}}$ can be chosen to be sufficiently small such that the distance $p(n) \cdot \delta^{\text{tiny}}$ is smaller than $1/q(n)$, establishing that any two points in the sequence $\mathbf{y}_1, \ldots, \mathbf{y}_{p^C}$ must lie in the same color-region or adjacent color-regions. ∎

*Observations on $F$.*

- We call $F$ a Borsuk–Ulam-style function—The suffix "style" is to note that we define a kind of function that has desirable properties similar to those of a Borsuk–Ulam function, but, for example, the domain of the function is $D$ as opposed to a sphere. Also, the function $F$ is "approximately Lipschitz" rather then truly continuous, which is good enough for our purposes.
- $|F(\mathbf{x})| \leq \varepsilon$ (here, $|F|$ denotes the $L_\infty$ or "maximum" norm of $F$) iff $\mathbf{x}$ encodes an approximate Consensus-Halving solution. Regarding this point, $F$ is not simulating a Borsuk–Ulam function, but rather simulating a function consisting of the difference between the values taken by a Borsuk–Ulam function, at two antipodal points.

### 6.2. Encoding the output of $F$ with a Consensus-Halving solution.

Proposition 6.7. *Let $S_{CH}$ be an $\varepsilon$-approximate solution to $I_{CH}$. Suppose that the c-e cuts of $S_{CH}$ represent a point $\mathbf{x}$ for which all points within distance $\delta^{\text{tiny}}$ of $\mathbf{x}$ lie in the Significant Region. Then we can reconstruct a solution to $I_{VT}$ in polynomial time.*

Recall that $S_{CH}$, $I_{CH}$, and $I_{VT}$ and $\varepsilon$ are as introduced at the start of section 6.

*Proof.* The general idea is that most circuit-encoders correctly output the color of a color-region in the vicinity of $\mathbf{x}$, and in order for us to have an approximate solution, two of these colors must be equal and opposite and represent adjacent color-regions in the twisted tunnel having opposite colors.

Observation 6.4 tells us that if circuit-encoder $C_i$ receives reliable inputs, it outputs the color of a point in the Möbius-simplex that lies within $\delta^{\text{tiny}}$ of $\mathbf{x}$.

The feedback received by the c-r agents in $I_{CH}$ corresponds to the average (over $i \in [p^C]$) of the feedback received by the individual circuit-encoders $C_i$. In detail, the $i$th coordinate of a typical point in $B = [-1, 1]^n$ (the vector of error values of the c-r agents) is obtained by taking c-r agent $a_i$ and (given any attempt at a consensus-halving solution $S$) subtracting $a_i$'s value for the parts of the consensus-halving domain labelled $A_-$ according to $S$, from those labelled $A_+$. The resulting point is at the center (or origin) of $B$ iff the c-r agents have balanced allocations of $A_+$ and $A_-$ (as required for a consensus-halving solution), and more generally, a point

in $[-1,1]^n$ is close to the center of $B$ iff the c-r agents have approximately balanced allocations of $A_+$ and $A_-$.

Observation 4.11 tells us that at most $n$ circuit-encoders fail to receive reliable input. For each c-r agent, its total discrepancy between $A_+$ and $A_-$ is at most $\varepsilon$ at a solution. Among all $p^C$ circuit-encoders, at most $n$ of them receive unreliable input, so if we sum over the reliable circuit-encoders, the total discrepancy should be at most $\varepsilon + \frac{n}{p^C}$. The value of $F$ (as defined in subsection 6.1) needs to be within distance $\varepsilon + \frac{n}{p^C}$ of the center of $B$.

We argue that to keep the error to at most $\varepsilon + \frac{n}{p^C}$ (in absolute value) in each coordinate, some of the circuit-encoders that receive reliable input must in fact give equal-and-opposite outputs (that can accordingly be easily computed from $S_{CH}$).

If we sum the feedback vectors from the reliable-input circuit-encoders and there are no cancellations, then each vector contributes at least $1/p^C$ to the sum of the absolute values of the entries of their total. Some entry of this total must get absolute value at least $\frac{1}{n}\alpha$, where $\alpha \approx 1$ is the fraction of circuits that have reliable inputs. This quantity is larger than $\varepsilon + \frac{n}{p^C}$, so there must be cancellations. But, could those cancellations involve a vector $\mathbf{x}_i$ that lies in the Significant Region but outside the twisted tunnel? The answer is no, from the way $f$ was defined (see Definition 5.7 and Figure 18) on the outer regions: the colors of any point in the outer regions are chosen to avoid being opposite to any color of any point of an adjacent color-region. We also use Proposition 6.6, telling us that points within distance $\delta^{\text{tiny}}$ of each other must come from the same color-region or adjacent color-regions. To conclude, some of the $p^C$ points represented by $S_{CH}$ must come from color-regions in the twisted tunnel having opposite colors, representing a solution to $I_{VT}$. ∎

It remains to rule out the possibility of a "solution" $\mathbf{x}$ existing at a greater distance from the twisted tunnel.

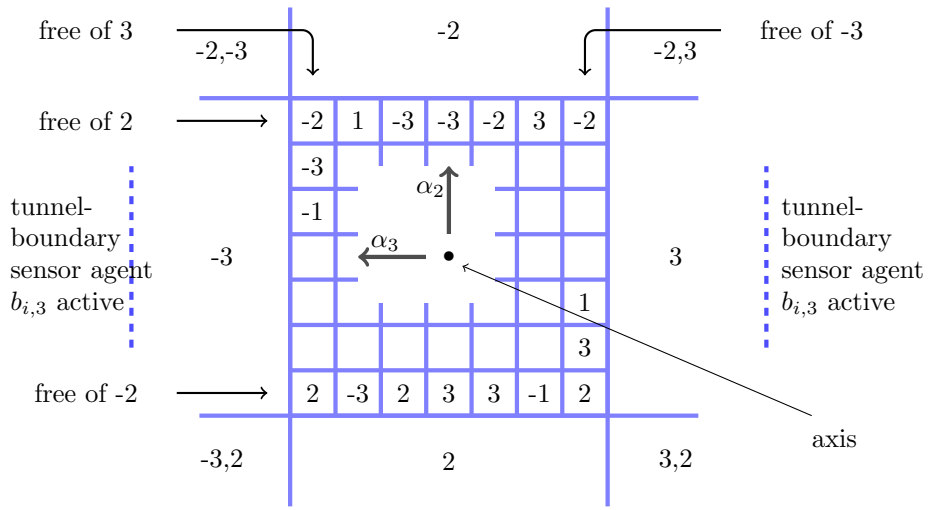


FIG. 18. *Cross section of the twisted tunnel (for $n = 3$), with examples of possible labels of regions. Note that the outer regions of the Significant Region are not adjacent to any cubelet having an opposite color to that outer region. For example, the left-hand column is free of cubelets with color 3 and is adjacent to the outer region with color $-3$.*

**6.3. No bogus approximate-zeroes of $F$ at boundary of Significant Region.** Proposition 6.7 tells us that $\varepsilon$-approximate zeroes of $F$ (inputs for which $F$ has value in $[-\varepsilon,\varepsilon]^n$) within the twisted tunnel $T$ must encode solutions. Around $T$, there are outer regions $R(S)$; note that if $i \in S$, then $-i \notin S$ and, moreover, there is an inverse-polynomial lower bound on the distance between any pair of points belonging to outer regions containing opposite colors. But we have to rule out points with color $j \in S$ being averaged with nearby points that are "colored" $-j$ due to a tunnel-boundary sensor agent. In more detail, if $\mathbf{x} \in R(S)$ and $\mathbf{x}$ is within $\delta^{\text{tiny}}$ of $\mathbf{x}'$ for which the $j$th tunnel-boundary sensor agent is active and provides feedback corresponding to $-j$, then we will prove that $S$ contains some other color $k \neq j$ and no point in a $\delta^{\text{tiny}}$-neighborhood of $\mathbf{x}$ activates the $k$th tunnel-boundary sensor $b_{1,k}$ to provide feedback corresponding to $-k$.

In the following, we will refer to cuts in the following manner: "cut $i$" refers to the $i$th cut (from left to right) in the c-e region. Also, recall that the width $\delta^T$ of the twisted tunnel is smaller than any other inverse-polynomial quantities of interest, apart from $\delta^{\text{tiny}}$, which itself is smaller than all other inverse-polynomials of interest, including $\delta^T$. We provide the following definition of a *consistent color*.

DEFINITION 6.8 (consistent colour). *For $\mathbf{x} = (\tau;\alpha_2,\dots,\alpha_n)$ in the Significant Region, color $j \in \{\pm 2,\dots,\pm n\}$, let $A_j \in \{A_+, A_-\}$ be the label that tends to increase in interval $[j-2, j]$ when the $|j|$th coordinate $\alpha_j$ of $\mathbf{x}$ is increased if $j > 0$ or decreased if $j < 0$. ($A_j$ depends on the sign and parity of $j$.) We say that $\mathbf{x}$ has* consistent color *$j$ if*

1. *if $j > 0$, then $\alpha_j > 2\delta^T$; if $j < 0$, then $\alpha_{|j|} < -2\delta^T$;*
2. *at least $\frac{1}{2} - \frac{p^{\text{large}}}{2p^{\text{huge}}}$ of the interval $[j-2, j]$ gets the label $A_j$.*

Item 1 says that at $\mathbf{x} \in R(S)$, color $j$ is a member of $S$ and the corresponding transformed coordinate is sufficiently far from the twisted tunnel. Item 2 says that we are at least some (small but significant) distance from triggering the $j$th tunnel-boundary sensor in a direction that corresponds to excessive colour $-j$. In other words, for the $j$th tunnel-boundary sensor to become active in direction $-j$, we would have to increase $A_{-j}$ by an inverse-polynomial amount.

The following proposition establishes that for points in the outer regions $R(S)$, consistent colors exist.

PROPOSITION 6.9. *Suppose $\mathbf{x} = (\tau;\alpha_2,\dots,\alpha_n)$ belongs to outer region $R(S)$ and that $\mathbf{x}$ is within distance $\delta^{\text{tiny}}$ of the boundary of the Significant Region. Then $\mathbf{x}$ has a consistent color in $\{\pm 2,\dots,\pm n\}$.*

*Proof.* Let $\ell \in \arg\max_{i\in\{2,\dots,n\}} |\alpha_i|$ be the index of a transformed coordinate with maximum absolute value. We may assume there is an inverse-polynomial quantity $\delta^+$ such that for any point $\mathbf{x} = (\tau;\alpha_2,\dots,\alpha_n)$ within distance $\delta^{\text{tiny}}$ of the boundary of the Significant Region, we have $|\alpha_\ell| \le \delta^+$. Moreover, the width $\delta^T$ of the twisted tunnel is chosen to be much smaller that $\delta^+$ (by an inverse-polynomial amount) but much larger than $\delta^{\text{tiny}}$ (by an inverse-polynomial amount), as explained earlier. Let

$$j = \begin{cases} \ell & \text{if } \alpha_\ell > 0, \\ -\ell & \text{if } \alpha_\ell < 0, \end{cases}$$

and let $\delta = |\alpha_\ell|$, so $\mathbf{x}$ is displaced distance $\delta > 0$ from the axis in direction $d_j^\tau$. Recall that $A_j \in \{A_+, A_-\}$ denotes the label that increases in the c-e region when we move in direction $d_j^\tau$. Also, let $A_{-j} \in \{+,-\}$, $A_{-j} \neq A_j$, be the complementary label. For

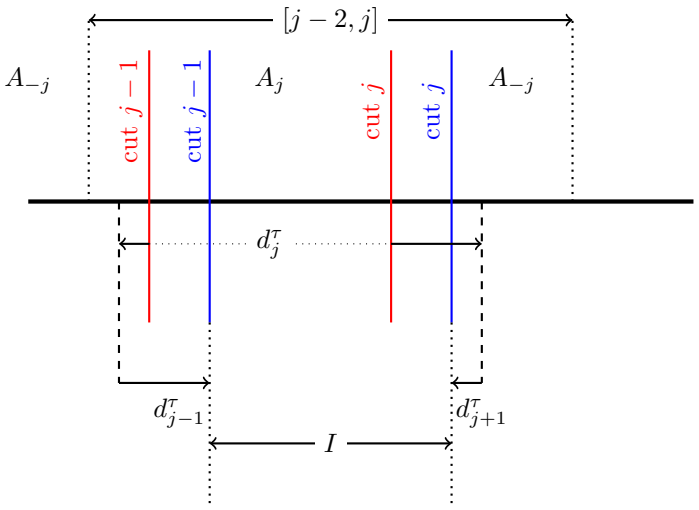


FIG. 19. *Illustration for Proposition* 6.9, *Case* 1. *For $i \in [n]$, cut $i$ denotes the $i$th (c-e) cut from the left. The cuts $j-1$ and $j$ colored red correspond to positions encoding part of $\mathbf{0}_\tau$. The dashed lines (to the left and to the right of the positions of the red cuts, respectively) correspond to positions encoding part of point $\mathbf{x}_j$, after we have only moved $\alpha_j$ in direction $d_j^\tau$. The cuts colored blue correspond to positions resulting from subsequent movement in directions $d_{j-1}^\tau$ and $d_{j+1}^\tau$, which encode part of $\mathbf{x}$. In the figure, the case where the average of the movements in $d_{j-1}^\tau$, $d_j^\tau$, and $d_{j+1}^\tau$ forces both cuts to move to the right, compared to their original positions in the encoding of $\mathbf{0}_\tau$, is shown.*

$j > 0$, this involves cuts $j$ and $j-1$ moving away from each other; for $j < 0$, this involves them moving toward each other. We consider two main cases, depending on the sign of $j$.

*Case* 1: $j > 0$ (i.e., $\alpha_j > 0$). In this case, moving in direction $d_j^\tau$ causes cuts $j-1$ and $j$ to move away from each other; this is illustrated in Figure 19.

We claim that $j$ is a consistent color for $\mathbf{x}$. Note first that $\alpha_j > 2\delta^T$ and therefore item 1 is satisfied, since $j > 0$ in this case. $\alpha_j > 2\delta^T$ follows from the fact that $j \in \arg\max_{i\in\{2,\dots,n\}} |\alpha_i|$, we are close to the boundary of the Significant Region, and the width of Significant Region is polynomially larger than that of the twisted tunnel. In order to be close to the boundary of the Significant Region, we must have moved more than $2\delta^T$ in some direction from $\mathbf{0}_\tau$ and by the choice of $j$, it holds that $\alpha_j > 2\delta^T$.

For item 2, recall first that $\mathbf{x}$ has transformed coordinates $(\tau; \alpha_2, \dots, \alpha_n)$ and that the origin of $D_\tau$ has transformed coordinates $\mathbf{0}_\tau = (\tau; 0, \dots, 0)$. The $(j-1)$st and $j$th cuts corresponding to the point $\mathbf{0}_\tau$ are located at positions $j-2+\tau$ and $j-1+\tau$, respectively, and are shown in red in Figure 19. Near the axis, where the cuts are evenly spaced (see Proposition 4.15), movement in direction $d_j^\tau$ corresponds to moving the $(j-1)$st and $j$th cuts (in the c-e region) away from each other. We will consider moving from $\mathbf{0}_\tau$ to $\mathbf{x}$ via a point $\mathbf{x}_j$ in which we will only have increased the transformed coordinate $\alpha_j$.

First, consider moving from $\mathbf{0}_\tau$ to point $\mathbf{x}_j = (\tau; 0, \dots, 0, \alpha_j, 0, \dots, 0)$ for $\alpha_j > 0$. In this process, we move the $(j-1)$st cut to the left by $\alpha_j \cdot \tau$ and the $j$th cut to the right by $\alpha_j \cdot (1-\tau)$; all this takes place within the interval $[j-2, j]$ (see Figure 19). Now consider moving from $\mathbf{x}_j$ to $\mathbf{x}$. In this process, the $(j-1)$st cut

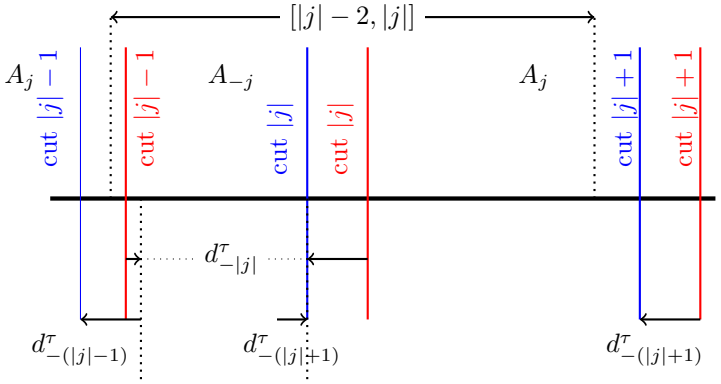


FIG. 20. *Illustration for Proposition* 6.9, *Case 2b(i). In this case, $j$ is negative and therefore we use $|j|$ to represent the $j$-th cut from the left. Moving in direction $d^\tau_j$ causes cuts $|j|-1$ and $|j|$ to move towards each other. Again, the red cuts correspond to part of $\mathbf{0}_\tau$ and the blue cuts correspond to the encoding of part of $\mathbf{x}$, after averaging over the movement in directions $d^\tau_{-(|j|-1)}$, $d_j = d_{-|j|}$ and $d^\tau_{-(|j|+1)}$. The figure shows a case where cut $|j|-1$ has moved outside the interval $[|j|-2,|j|]$ to the left, in which case the whole subinterval $[|j|-2, c(|j|)]$ (the interval between $|j|-2$ and the position of the blue cut $|j|$) receives the label $A_{-j}$. Note however that cut $|j|+1$ does not intersect the interval $[|j|-2,|j|]$ and therefore there is no additional amount of $A_{-j}$ introduced to the RHS of $[|j|-2,|j|]$, therefore we are in Case 2b(i). The increase of $A_{-j}$ due to the movement of cut $|j|-1$ to the left is entirely compensated by the decrease of $A_{-j}$ because of the movement of cut $|j|$ to the left.*

moves to the right by $\alpha_{j-1}\cdot(1-\tau)$ and the $(j+1)$st cut moves to the left by $\alpha_{j+1}\cdot\tau$. From the choice of $j$ to be $j \in \arg\max_{i\in\{2,\dots,n\}} |\alpha_i|$, it follows that $\alpha_{j-1} \le \alpha_j$ and $\alpha_{j+1} \le \alpha_j$. Then, there is a subinterval of $[j-2, j]$ that contains the unit-length interval $I = [j-2+\tau+\alpha_j\cdot(1-2\tau), j-1+\tau+\alpha_j\cdot(1-2\tau)]$, which ends up colored entirely $A_j$, implying item 2. Overall, we obtain that $j$ is a consistent color.

*Case* 2: $j < 0$ (i.e., $\alpha_j < 0$). In this case, moving in direction $d^\tau_j$ causes cuts $|j|-1$ and $|j|$ to move toward each other; this is illustrated in Figure 20.

*Case* 2a: $\tau \in [1/2n, 1-(1/2n)]$. In this case, all movements of the cuts, in and around the Significant Region, are in distances upper-bounded by $\delta^w$, which by Proposition 4.15 is smaller than $1/2n$ by an inverse-polynomial amount. This means that if we start at $\mathbf{0}_\tau$ and reset individual transformed coordinates to those of $\mathbf{x}$, in any order (i.e., going through any intermediate point $\mathbf{x}_j$, similarly to above), the movement of the cuts will never force them to cross integer-valued thresholds. In other words, in moving from $\mathbf{0}_\tau$ to $\mathbf{x}$, only the relevant cuts $j-1$ and $j$ will lie in the interval $[j-2, j]$. This case can be seen in the illustration of Figure 19 if one reverses the direction of the arrows, switches the labels $A_j$ to $A_{-j}$ and vice versa, and substitutes $j$ by $|j|$ in the labelling of cuts. The argument establishing the existence of a consistent color is exactly symmetric to that of Case 1 above.

*Case* 2b: $\tau \in [0, 1/2n] \cup [1-(1/2n), 1]$. Here, we consider the case where $\tau \in [0, 1/2n]$; the other case is similar by symmetry. This case is illustrated in Figure 20; note that the sequence of labels $A_j/A_{-j}$ is switched to make $A_j$ the label that increases when we move in direction $d^\tau_j$.

Moving in direction $d^\tau_j$ causes an increase of the label $A_j$ in the interval $[|j|-2, |j|]$. For $j$ not to be a consistent color, we should observe an excess of the label $A_{-j}$ in

this interval. In generating cut locations from coordinates of $\mathbf{x}$, the amount of $A_{-j}$ in $[|j|-2,|j|]$ can be raised in the following ways (see Figure 20):

- By increasing the transformed coordinate $\alpha_{|j|-1}$ in the negative direction, moving in direction $d^{\tau}_{-(|j|-1)}$. This causes cuts $|j|-2$ and $|j|-1$ to move toward each other and therefore importantly for us here, cut $|j|-1$ to move to the left (toward integer point $|j|-2$).
- By increasing the transformed coordinate $\alpha_{j+1}$ in the negative direction, moving in direction $d^{\tau}_{-(|j|+1)}$. This causes cuts $|j|$ and $|j|+1$ to move toward each other.

Note that what may happen in this last case is that cut $|j|+1$, which used to lie to the right of the integer point $|j|+2$ before moving in direction $d^{\tau}_{-(|j|+1)}$, now lies to the left of the integer point $|j|+2$ after the movement, therefore increasing the label $A_{-j}$ at the *RHS* of $[|j|-2,|j|]$. We consider two more cases, depending on whether or not this is the case.

*Case* 2*b*(i): At $\mathbf{x}$, cut $|j|+1$ is to the right of location $|j|$ or at location $|j|$.

There are two ways to restore the deficit of $A_{-j}$ that resulted from moving in direction $d^{\tau}_{j}$ from $\mathbf{0}_{\tau}$ to $\mathbf{x}_j$. Moving in direction $d^{\tau}_{-(|j|-1)}$ moves cut $|j|-1$ to the left, and moving in direction $d^{\tau}_{-(|j|+1)}$ moves cut $|j|$ to the right. (Note that the movement of cut $|j|+1$ to the left has not changed the balance of $A_j$ and $A_{-j}$ in the interval $[|j|-2,|j|]$ any further, by the assumption of the case.) Since $j$ was chosen to be in $\arg\max_{i\in\{2,\ldots,n\}}|\alpha_i|$, it is easy to verify the following:

- Cut $|j|-1$ has moved to the left as a result of moving in direction $d^{\tau}_{-(|j|-1)}$ *at most as much* as cut $|j|$ has moved to the left as a result of moving in direction $d^{\tau}_{j}$ (from $\mathbf{0}_{\tau}$ to $\mathbf{x}_j$).
- Cut $|j|$ has moved to the right as a result of moving in direction $d^{\tau}_{-(|j|+1)}$ *at most as much* as cut $|j|-1$ has moved to the right as a result of moving in direction $d^{\tau}_{j}$ (from $\mathbf{0}_{\tau}$ to $\mathbf{x}_j$).

Therefore, a large enough subinterval of $[|j|-2,|j|]$ has been colored with $A_j$, which means $j$ is a consistent color.

*Case* 2*b*(ii): At $\mathbf{x}$, cut $|j|+1$ is to the left of location $|j|$. See Figure 21.

In this case, we have $\alpha_{|j|+1}<0$ and movement in direction $d^{\tau}_{-(|j|+1)}$ causes cuts $|j|$ and $|j|+1$ to move toward each other. Note also that besides the effect of the movement in direction $d^{\tau}_{-(|j|+1)}$, cut $|j|+1$ may move to the left due to movement in direction $d^{\tau}_{|j|+2}$, since such a movement would cause cuts $|j|+1$ and $|j|+2$ to move away from each other, and therefore, cut $|j|+1$ to move to the left. However, the distance moved in direction $d^{\tau}_{|j|+2}$ is small; it is at most $\tau\cdot|\alpha_j|$, which is at most $\tau\cdot\delta^{+}$. Therefore, we need movement at least $\tau(1-\delta^{+})$ in direction $d^{\tau}_{-(|j|+1)}$ in order to cover the distance moved in direction $d^{\tau}_{j}$.

First, we verify that $-(|j|+1)$ satisfies item 1 of Definition 6.8, i.e., that $\alpha_{|j|+1}<-2\delta^{T}$ (at this point we know that $\alpha_{j+1}$ is a negative quantity). We consider two cases, depending on whether $\tau$ is "small" or "large" (relatively to the small interval $[0,1/2n]$).

- In the case when $\tau<\frac{1}{4}|\alpha_j|$, the largest part of the deficit of $A_{-j}$ introduced by moving from $\mathbf{0}_{\tau}$ to $\mathbf{x}_j$ results from moving cut $|j|$ to the left. However, letting $c(|j|-1)$ denote the position of cut $|j|-1$ after this movement, the interval $[|j|-2,c(|j|-1)]$ is too small for the movement of cut $|j|-1$ in

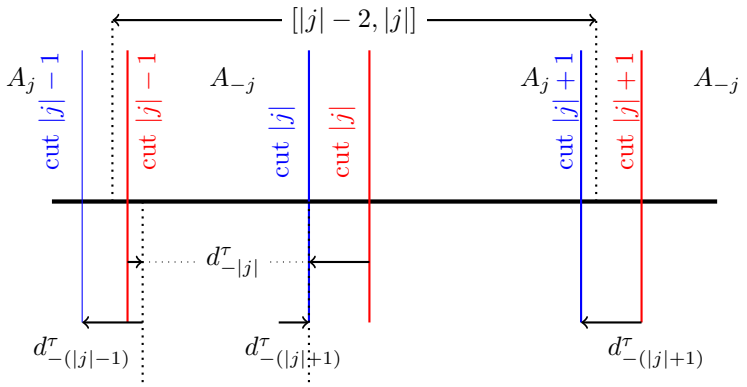


FIG. 21. *Illustration for Proposition* 6.9*, Case* 2b(ii)*. In this case, $j$ is negative and therefore we use $|j|$ to represent the $j$th cut from the left. Moving in direction $d^\tau_j$ causes cuts $|j|-1$ and $|j|$ to move toward each other. Again, the red cuts correspond to part of $\mathbf{0}_\tau$ and the blue cuts correspond to the encoding of part of $\mathbf{x}$, after averaging over the movement in directions $d^\tau_{-(|j|-1)}$, $d_j = d_{-|j|}$, and $d^\tau_{-(|j|+1)}$. The figure shows a case where cut $|j|-1$ has moved outside the interval $[|j|-2,|j|]$ to the left, in which case the whole subinterval $[|j|-2, c(|j|)]$ (the interval between $|j|-2$ and the position of the blue cut $|j|$) receives the label $A_{-j}$. Additionally, cut $|j|+1$ has moved to the left and now intersects the interval $[|j|-2,|j|]$ introducing an additional amount of $A_{-j}$ to the RHS of $[|j|-2,|j|]$. By the argument of Case* 2b(ii)*, either $-(|j|+1)$ will be a consistent color or there will be some interval $[l-2,l]$ ($l>0$, possibly $[n-2,n]$) for which the overlap between $[l-2,l]$ and the cut $l+1$ will be bounded by $p^{\text{large}}/2p^{\text{huge}}$ and we will have a consistent color; see also Figure* 22.

direction $d^\tau_{-(|j|-1)}$ to compensate. In other words, even if movement in direction $d^\tau_{-(|j|-1)}$ moves cut $|j|-1$ to the left endpoint of the interval $[|j|-2,|j|]$, this is not enough to make up for the deficit of $A_{-j}$ introduced from the movement in direction $d^\tau_j$. This means that cut $|j|+1$ needs to move to the left as well, and in particular, it needs to move by more than $\tau/4$ to the left of location $j$. This is only possible if $\alpha_{|j|+1} < -2\delta^T$.

- In the case when $\tau \geq \frac{1}{4}|\alpha_j|$, since $\tau$ is large enough, cut $|j|+1$ needs to move a substantial distance to the left, in order to end up positioned to the left of integer position $|j|$. In particular, it needs to move at least $\frac{1}{4}|\alpha_j| - \tau \cdot \delta^+$ to the left. This implies that item 1 is satisfied for color $-(|j|+1)$.

Now consider what needs to happen in order for the second condition to fail. Consider the interval $[|j|-1,|j|+1]$ (which is monitored by the $(j+1)$st tunnel-boundary sensor). Since cut $|j|+1$ is located to the left of location $|j|$ (the midpoint of this interval), there exists a subinterval of length at most 1 labelled $A_j$, within $[|j|-1,|j|+1]$. This means that either

- the color $-(|j|+1)$ is a consistent color and we are done or
- there is an additional amount of label $A_j$ within interval $[|j|-1,|j|+1]$ and the total number of value-blocks labelled $A_j$ outnumbers that of those labelled $A_{-j}$ by at least $p^{\text{large}}$. The only way this can happen is if cut $|j|+2$ lies to the left of the integer location $|j|+1$, and in fact, it has to lie an inverse-polynomial distance, at least $\frac{p^{\text{large}}}{2p^{\text{huge}}}$, to the left of $|j|+1$.

In the case when that happens, we move on to consider interval $[j,j+2]$ and we apply the same argument. Again, $\alpha_{j+2}$ is negative, and since cut $|j|+2$ is to the

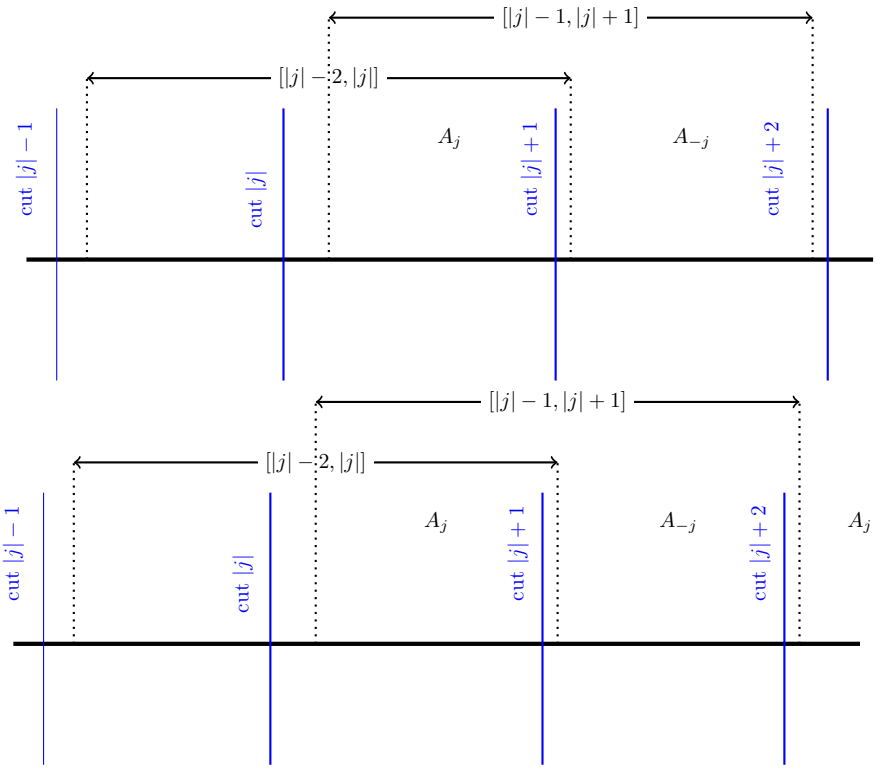


FIG. 22. *Illustration for the two cases in the last argument of Proposition* 6.9*, Case* 2b(ii)*. At the top, there is only one sub-interval labelled* $A_j$ *in the interval* $(|j|+1,|j|+1)$*, and colour* $(-|j|+1)$ *is a consistent colour. At the bottom, the label* $A_j$ *appears again on the RHS of the interval* $(|j|+1,|j|+1)$*, as a result of the cut* $|j|+2$ *moving to the left of the integer point* $|j|+1$*. In that case, the same argument is applied to the interval* $(|j|,|j|+2)$*.*

left of location $j+1$ by a margin $\frac{p^{\text{large}}}{2p^{\text{huge}}} < 2\delta^T$, $-(|j|+2)$ satisfies item 1 to be a consistent color. It will also satisfy item 2, unless cut $|j|+3$ lies to the left of location $|j|+2$ by an inverse-polynomial distance, at least $\frac{p^{\text{large}}}{2p^{\text{huge}}}$, similarly to before.

Continuing like this, we will either find a consistent color in some interval $[j-2,j]$ with $j<n$, or we will reach interval $[n-2,n]$. When we reach interval $[n-2,n]$, cut $n$ has had to move to the left of integer location $n-1$ in order to prevent $-(n-1)$ from being a consistent color (as otherwise we would have identified a consistent color in some already examined interval). But then $-n$ is a consistent color, since we have moved an inverse-polynomial distance (at least $\frac{p^{\text{large}}}{2p^{\text{huge}}} < 2\delta^T$) in direction $d^{\tau}_{-n}$ (item 1), and at least $1/2$ of the interval $[n-2,n]$ is colored in a way that agrees with this (item 2). ∎

COROLLARY 6.10. *A solution* $S_{CH}$ *to* $I_{CH}$ *cannot encode a point* $\mathbf{x}$ *in the Significant Region, within distance* $\delta^{\text{tiny}}$ *of the boundary (where tunnel-boundary sensor agent(s) become active).*

*Proof.* Observation 6.4 tells us that the $k$th component of $f'$ is the difference between $A_+$ and $A_-$ observed by c-r agent $a_k$, and Proposition 6.5 tells us that all

these components, averaged over a set of points within $\delta^{\text{tiny}}$ of $\mathbf{x}$, need to be close to zero, at a solution.

Proposition 6.9 tells us that $\mathbf{x}$ has some consistent color $k$. All points within $\delta^{\text{tiny}}$ of $\mathbf{x}$ cause two outputs (gates $g'_k$, $g'_{-k}$ as defined in subsection 5.1.2) of the circuit-encoders to represent color $k$. This includes points where tunnel-boundary sensor agents are active, since by the properties of consistent colors, we are at least an inverse-polynomial distance from any point where any $b_{i,k}$ can be active in the wrong direction. In subsection 5.1.3 the tunnel-boundary sensor agents are designed to agree with the definition of consistent color, Definition 6.8. So c-r agent $a_k$ observes a large imbalance between $A_+$ and $A_-$. ∎

### 6.4. No bogus approximate-zeroes of $F$ due to the connecting facet.

PROPOSITION 6.11. *Let* $\mathbf{x}$, $\mathbf{x}'$ *be points in the Significant Region having transformed coordinates* $(\tau; \alpha_2, \ldots, \alpha_n)$ *and* $(1-\tau; -\alpha_2, \ldots, -\alpha_n)$, *respectively, for* $\tau < \frac{1}{2} - \delta^T$. *Then* $f(\mathbf{x}) = -f(\mathbf{x}')$.

*Proof.* The proposition extends Observation 5.5. The points $\mathbf{x}$ and $\mathbf{x}'$ have been colored according to item 2 of subsection 5.3, and they belong to two long thin color-regions that extend the cubelets that lie on the panchromatic facets of the cube embedded at the center of $T$, all the way to the ends of $T$. From the boundary conditions on the colouring of box $B$ in NEW VARIANT HIGH-D TUCKER, and the way $f$ is constructed above, their colors are equal and opposite. ∎

*Remark.* The $\mathbf{x}$, $\mathbf{x}'$ in Proposition 6.11 will "approach each other" as $\tau \to 0$. That is, they correspond to sequences of CONSENSUS-HALVING cuts where the left-hand cut in the c-e region "wraps around" to the RHS of the c-e region. Proposition 6.11 may thus seem to create Borsuk–Ulam directions that are in conflict with each other as we cross from facet $D_0$ to $D_1$, but in fact the flip of labels in CONSENSUS-HALVING that occurs when we move from $D_0$ to $D_1$ will mean that they are in agreement with each other.

We consider the case where the set of $p^C$ points in $D$ represented by the solution $S_{CH}$ to $I_{CH}$ contains points on opposite sides of the facets of $D$ that have been identified with each other. Proposition 6.11 tells us that color-regions are adjacent to color-regions having the opposite color. We need to verify that for a pair $\mathbf{x}, \mathbf{x}'$ of points that are close together but have opposite colors (due to lying in such a pair of color-regions) the same (and not opposite) feedback is provided to the c-r agents (in contrast with a pair of opposite-color points that represent a solution, whose feedback to the c-r agents cancel each other out).

In reasoning about these elements $\mathbf{x}, \mathbf{x}' \in D$, it is helpful to depart from our convention that the label-sequence begins with $A_+$, and suppose that for $\mathbf{x}'$, the label-sequence begins with $A_-$. Suppose $\mathbf{x}, \mathbf{x}'$ have corresponding circuit-encoders $C_i, C_{i'}$ and assume that $C_i$ and $C_{i'}$ receive reliable inputs, recalling that only $n$ circuit-encoders may fail to receive reliable inputs. Notice that if $\mathbf{x}$ causes a tunnel-boundary sensor agent $b_{i,j}$ to be active in direction $A_+$, then $\mathbf{x}'$ typically causes $b_{i',j}$ to be active in direction $A_+$ also (the overrepresented label is fed back to c-r agent $a_j$).

In the case that no tunnel-boundary sensor agents are active, if $\mathbf{x}, \mathbf{x}'$ receive opposite colors from $C_i, C_{i'}$, then, reverting to our convention that the shared label-sequence begins with $A_+$, we note that their reference-sensor agents get opposite labels, which causes $C_i$ and $C_{i'}$ to agree with each other.

*Remark.* For intuition, it is possibly helpful to think about the move from $\mathbf{x}$ to $\mathbf{x}'$ in terms of operations on the coordinate-encoding cuts. At $\mathbf{x}$, there is a cut on

TABLE 1
*An inventory of the most crucial parts of the reduction and where they can be found within the text.*

| | |
|---|---|
| 2D-TUCKER is PPA-complete | [2] |
| VARIANT HIGH-D TUCKER is PPA-complete | Section 3 |
| Elements of the Consensus-Halving Instance | |
| General construction of the instance | Subsection 5.1 |
| c-e region, c-r agents, and c-e cuts | Definition 4.1, Definition 4.2, Remark 4.3 |
| How the c-r agents receive "feedback" from the circuit | Subsection 5.1.2 |
| Sensor agents | Definition 4.5 |
| How the sensor agents detect the positions of the cuts | Subsection 4.1, Figure 6 |
| Reference sensor agent | Definition 5.1 |
| Tunnel-boundary sensors agents | Definition 4.6 |
| When tunnel-boundary sensor agents are active | Definition 4.7 |
| How inactive tunnel-boundary sensors restrict the cuts in the c-e region | Proposition 4.15, Lemma 4.18, Lemma 4.19 |
| Effect of the tunnel-boundary sensors on the feedback mechanism | Subsection 5.1.3 |
| No bogus solutions due to the tunnel-boundary sensors | Subsection 6.3, Proposition 6.9 |
| Simulation of the TUCKER circuit | Subsection 4.1 |
| Preprocessing | Subsection 5.1.1 |
| The output gates | Subsection 5.1.2 |
| The Domain and the Coloring Function | |
| The Mobius-Simplex Domain | Definition 4.12 |
| How solutions to Consensus-Halving encode points in the domain | Subsection 4.2, Subsection 5.4 |
| The Significant Region | Definition 4.14 |
| An alternative coordinate system | Subsection 5.2 |
| Coordinate transformations in polynomial time | Subsection 5.2.1 |
| The twisted tunnel and the coloring of the domain | Definition 5.7 |
| Solutions to Consensus-Halving encode points in the twisted tunnel | Proposition 6.7 |
| No solutions outside the Significant Region | Subsection 6.3, Corollary 6.10 |
| No bogus solutions due to the connecting facet | Subsection 6.4, Proposition 6.11 |
| Robustness and Averaging | |
| Polynomially many copies | Subsection 4.1, Definition 4.4, Subsection 5.1.4 |
| Reliable input | Definition 4.10, Observation 4.11 |
| Reliable copies | Definition 6.2 |
| Stray cuts and their effects | Definition 6.1, Lemma 6.3 |

the RHS of the c-e region, and in moving to $\mathbf{x}'$ we move that cut to the LHS. If we move the cut while leaving the labels of the c-e region unchanged (apart from at the ends) we expect the circuit to behave as before, but since we have switched the roles of labels $A_+$ and $A_-$, the feedback to agents $a_1, \ldots, a_n$ gets inverted. We reinvert this feedback by reversing the colour, and hence the output of $f'$.

**7. PPA-completeness of** DISCRETE HAM SANDWICH**.** In this section, we show that DISCRETE HAM SANDWICH is PPA-complete.

**7.1. PPA-hardness.** As it happens, the PPA-hardness result for DISCRETE HAM SANDWICH follows rather straightforwardly, via a relatively simple reduction from NECKLACE-SPLITTING. The basic idea of Theorem 1.8 of embedding the necklace in the moment curve appears already in [72, 62] and [26, p. 48].

We reduce from 2-thief NECKLACE-SPLITTING, which is PPA-complete by Theorem 1.3. The idea is to embed the necklace into the *moment curve*, i.e., $\gamma = \{(\alpha, \alpha^2, \ldots, \alpha^n) : \alpha \in [0,1]\}$. Assume all beads lie in the unit interval $[0,1]$. A bead having color $i \in [n]$ located at $\alpha \in [0,1]$ becomes a point mass of ingredient $i$ of the ham sandwich located at $(\alpha, \alpha^2, \ldots, \alpha^n) \in \mathbb{R}^n$. It is known that any hyperplane intersects the moment curve $\gamma$ in at most $n$ points (e.g., see [62, Lemma 5.4.2]), and

therefore a solution to Discrete Ham Sandwich corresponds directly to a solution to Necklace-Splitting, where the two thieves splitting the necklace take alternating pieces. (In the $k = 2$ case, we may assume without loss of generality that they do in fact take alternating pieces.)

**7.2. Membership in PPA.** The recent work of [26] provides a proof of the Discrete Ham Sandwich theorem using Tucker's lemma, and we believe that containment in PPA is implicit in the proof. For completeness, we sketch a more explicit proof of containment in PPA, by reducing it to the version of Tucker's lemma of Freund and Todd [41]. This reduces the problem to Leaf since the proof in [41] constitutes a reduction from Tucker to Leaf (indeed, it is referenced in the PPA-membership result of the former problem in [71]).

Tucker's Lemma as stated in [41] is the following:

> *Let the vertices of a special triangulation $T$ of $B^n$ be assigned labels from $\{\pm 1, \ldots, \pm n\}$. If antipodal vertices of $T$ of $S^{n-1}$ receive complementary labels (labels that sum to zero), then $T$ contains a complementary 1-simplex (whose vertices have complementary labels).*

In the above statement, $B^n$ denotes the octahedral ball consisting of points in $n$-space within $L_1$-distance 1 from the origin. $S^{n-1}$ is the "surface" of $B^n$, points at $L_1$-distance 1 from the origin. A *special* triangulation is a centrally symmetric triangulation of $B^n$ that refines the octahedral subdivision (the hyperplanes normal to the axes that contain the origin).

In brief, [41] shows how to take such a vertex-labelled triangulation of $B^n$ and derive from it a "Leaf graph" $G$ as follows. The vertices of $G$ consist of certain simplices (or faces) of the triangulation. Define a sign vector to be an $n$-vector whose entries are from $\{0, 1, -1\}$; a sign vector indicates whether each coordinate of some point is 0, positive, or negative. Given a simplex $\sigma$ of a special triangulation, points in the relative interior of $\sigma$ all have the same sign vector $\mathrm{sgn}(\sigma)$. For a sign vector $s$, $\sigma$ is $s$-labelled if whenever $s_i$ is nonzero, $\sigma$ has a vertex labelled $i$, and we say that $\sigma$ is completely labelled if it is $\mathrm{sgn}(\sigma)$-labelled. Then the following hold:

- The vertices of $G$ correspond to completely labelled simplices of the triangulation (these are referred to as "happy simplices" in [63], which contains a nice exposition of Freund and Todd's proof).
- The presence of an edge between two vertices of $G$ indicates the following: one of the corresponding simplices is a facet of the other, and the smaller one has the same set of labels as the larger, or if both simplices lie on the boundary and are antipodal.

With all this in place, it follows that the graph $G$ (a) has degree at most 2 and (b) it has a specific vertex with degree 1, namely, the zero-dimensional simplex at the origin (which has the all-zeroes sign vector, hence is completely labelled). This implies the existence of a Tucker solution (a complementary 1-simplex) by a path-following argument, and the constructed graph is effectively an instance of Leaf.

To show that $n$-dimensional Discrete Ham Sandwich belongs to PPA, we will construct a special Tucker triangulation based on a set of (exponentially many) *candidate hyperplanes*—these hyperplanes will correspond to vertices of a special triangulation of the $(n+1)$-dimensional octahedral ball $B^{n+1}$ with labels that satisfy the Tucker property, so they can be converted to an instance of Leaf via the construction of Freund and Todd [41]. Since Theorem 1.8 reduces from 2-thief Necklace-Splitting to Discrete Ham Sandwich, it follows immediately that 2-thief Necklace-Splitting belongs to PPA. (In subsection 9.1 we go a bit further for Necklace-Splitting: we

show that NECKLACE-SPLITTING belongs to PPA whenever the number of thieves is a power of 2.)

Given an instance $I$ of DISCRETE HAM SANDWICH, we show how to construct a suitable triangulation. $I$ contains $n$ sets of points $\{S_1, \ldots, S_n\}$ in $n$-dimensional space; we assume coordinates are represented as fractions whose numerators and denominators are given via standard binary expansions. (Recall that we leave it as an open problem whether DISCRETE HAM SANDWICH remains PPA-hard for points presented in unary. Of course, the "in PPA" result follows immediately for that restricted version.) A hyperplane can be written as $\sum_i a_i x_i = b_i$, where the $x_i$ are the coordinates of a point in $n$-space. It corresponds with halfspaces $\sum_i a_i x_i \geq b_i$ and $\sum_i a_i x_i \leq b_i$. Suppose the halfspace $\sum_i a_i x_i \geq b_i$ is expressed as a vector $(a_1, \ldots, a_n, b)$ whose entries are assumed to be rescaled so that their absolute values sum to 1. Notice that $(a_1, \ldots, a_n, b)$ lies on the surface of the unit $L_1$-norm $n$-ball $B^{n+1}$. Given a halfspace, label it as follows. Let $S_i$ be the most unequally-split point set from $I$, breaking ties lexicographically. Label the halfspace with $+i$ if most of the points of $S_i$ belong to it; otherwise label it $-i$. Crucially, this labelling scheme has the property that $(a_1, \ldots, a_n, b)$ gets the opposite label to $(-a_1, \ldots, -a_n, -b)$.

Based on $I$, there exists some limited precision for numbers, over which we can search for a solution. In particular, letting $N$ be the product of the numerators and denominators of numbers occurring in $I$, there should be a solution corresponding to a halfspace $\frac{1}{N}(a_1, \ldots, a_n, b)$ where the $a_i$ and $b$ are integers whose absolute values sum to $N$. The candidate hyperplanes consist of all such vectors. Two candidate hyperplanes are adjacent in the triangulation of $B^{n+1}$ provided that one of their representative vectors is obtained from the other by subtracting $\frac{1}{N}$ from one entry and adding it to some other entry. We then add the origin to the set of vertices of the triangulation of $B^{n+1}$, give it the label $n+1$, and make it adjacent to all other points (corresponding to all the candidate hyperplanes). The resulting triangulation is special since it refines the octahedral subdivision (notice that a one-dimensional edge of the triangulation cannot connect a vertex with positive $i$th coordinate to one with negative $i$th coordinate). Also, antipodal vertices of $S^n$ receive opposite labels as required. The corresponding instance of LEAF as defined above has a degree-1 vertex that corresponds to the 0-simplex at the origin. Any other degree-1 vertex must correspond to a simplex in the triangulation containing a complementary 1-simplex, where two adjacent vertices (candidate hyperplanes) have opposite labels. The origin itself cannot form part of such a complementary 1-simplex since there is no other point with label $-(n+1)$, or indeed $n+1$. Any solution corresponds to two hyperplanes that differ incrementally, agree on some $S_i$ as the most unevenly split, but disagree on which side most points in $S_i$ lie. One of these hyperplanes must be a solution (recall Definition 1.7 allows us to break ties in our favor if a point lies on a hyperplane).

**8. Equivalence of Consensus-Halving and Necklace-Splitting.** In this section, we prove that approximate CONSENSUS-HALVING and NECKLACE-SPLITTING are computationally equivalent, in the sense that they reduce to each other in polynomial time; Theorem 1.6 follows from the results of this section. In fact, we will provide a stronger result, showing the equivalence between generalisations of the two problems, in which (a) the number of allowed cuts is $\ell$, for some $\ell$ which is bounded by a polynomial in $n$, and (b) the number of thieves in the Necklace Splitting instance and the number of labels in the Consensus Division instance is $k$ instead of 2. The former problem is the general Necklace Splitting problem studied by Alon [3], whereas the

latter problem was referred to as the approximate Consensus-$1/k$-division problem by Simmons and Su [75]. We define the problems formally below. Let $b_i(O)$ denote the number of beads in an interval $O$.

DEFINITION 8.1 ($(n, \ell, k)$-NECKLACE-SPLITTING).
- Input: *$k \cdot m$ beads placed on an interval $O$ with $\alpha_i \cdot k$ beads of color $i = 1, \ldots, n$, where $\alpha_i \in \mathbb{N}^+$, with $k \leq n$.*
- Output: *A partition of $O$ into $k$ parts $O_1, O_2, \ldots, O_k$ such that for each color $i = 1, \ldots, n$, it holds that for each $j \in \{1, \ldots, k\}$, it holds that $b_i(O_j) = \alpha_i$, using $(k-1) \cdot \ell$ cuts.*

DEFINITION 8.2 ($(n, \ell, \varepsilon)$-CON-$1/k$-DIVISION).
- *Input: The value measures $\mu_i : O \to R_+, i = 1, \ldots, n$, for $n$ agents and $k \leq n$.*
- *Output: A partition $(O_1, O_2, \ldots O_k)$ with $(k-1) \cdot \ell$ cuts such that $|u_i(O_t) - u_i(O_j)| \leq \varepsilon$ for all $t$ and $j$ and for all agents $i \in N$.*

We will use the terms "Consensus Division" and "Necklace-Splitting" loosely to refer to these problems without specifying the number of partitions or cuts.

**8.1. From approximate Consensus Division to Necklace-Splitting.** In this subsection, we will establish a reduction from $(n, \ell, \varepsilon)$-CON-$1/k$-DIVISION to $(n, \ell, k)$-NECKLACE-SPLITTING for all $\ell$ which are bounded by a polynomial in $n$. The following facts hold about any instance of $(n, \ell, \varepsilon)$-CON-$1/k$-DIVISION:
- All the agents' valuations are represented as piecewise constant functions.
- The number of pieces of these functions is upper bounded by some $p_{\mathcal{M}}(n)$ where $p_{\mathcal{M}}$ is a polynomial.
- The volume of each piece is upper bounded by some $p_{\mathcal{V}}(n)$ where $p_{\mathcal{V}}$ is a polynomial.

THEOREM 8.3. *The $(n, \ell, \varepsilon)$-CON-$1/k$-DIVISION problem is polynomial-time reducible to $(n, \ell, k)$-NECKLACE-SPLITTING, when the number of cuts $\ell$ is bounded by a polynomial in $n$ and $\varepsilon$ is inverse-polynomial in $n$.*

*Proof.* Let $\mathcal{C}$ be such an instance of $(n, \ell, \varepsilon)$-CON-$1/k$-DIVISION. We will construct an instance $\mathcal{B}$ of $(n, \ell, k)$-NECKLACE-SPLITTING as follows: For each agent $i \in \{1, 2, \ldots, n\}$ of $\mathcal{C}$, the following hold:
- Associate a different color $c_i$.
- Repeat for all of agent $i$'s valuation blocks:
  - Let $V$ be the volume of the block and let $\alpha$ be the interval on which the block is defined. Divide the block into $\lceil V/\delta \rceil$ subblocks of volume $\delta$ each, where
$$\delta = \frac{\varepsilon}{n^3[(k-1)(\ell+1) + p_M(n)]},$$
except possibly the last subblock, which will have volume $\delta' \leq \delta$. We will call such a subblock an *imperfect subblock*. Let $\alpha_j$ denote the corresponding intervals for $j = 1, \ldots, \lceil V/\delta \rceil$.
  - Place a bead of color $c_i$ in the middle of each interval $\alpha_j$.
- If the total number $b_i$ of beads of color $c_i$ placed is not a multiple of $k$, remove $b_i \bmod k$ beads of color $c_i$. We will refer to these beads as the *parity beads*.

Intuitively, each bead "represents" a valuation block of volume $\delta$ and some beads represent the imperfect subblocks of smaller volume. See Figure 23 for an example of the construction, when $k = 2$ and $\ell = n = 2$. Note that the construction requires to

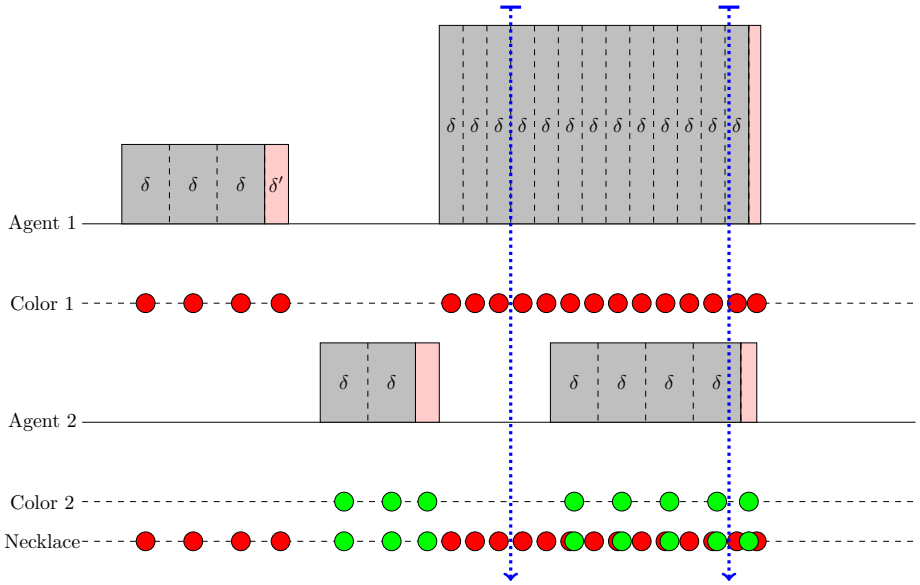


FIG. 23. *An example of the reduction when $k = 2$ and $\ell = n = 2$. The red (dark) beads correspond to Agent 1 and the green (light) beads correspond to Agent 2. The pink (lightgray) area corresponds to the last part of each valuation block, which has volume $\delta' \leq \delta$. The blue dotted lines indicate the positions of the cuts; note that while the first cut lies exactly at the boundary of two subblocks of valuation $\delta$ for Agent 1 and in a "value-free" region for Agent 2, the second cut intersects the interior of a subinterval for both agents.*

partition the instance into at most $p_{\mathcal{M}}(n) \cdot p_{\mathcal{V}}(n)/\delta$ intervals and find their midpoints and therefore runs in polynomial time. Next, we will argue for correctness.

Let $\mathcal{S}$ be a solution to $\mathcal{B}$ (which uses $(k-1)\ell$ cuts); we will prove that having the same cuts in the same positions precisely gives a solution to $(n, \ell, \varepsilon)$-CON-$1/k$-DIVISION. Consider any agent $i$ and label the beads of color $c_i$ with $j = 1, \ldots, t$ for some $t \in \mathbb{N}$ being the total number of beads of color $c_i$ according to the construction above. Let $[d_j, d_{j+1}]$ be the interval defined by two consecutive such beads.

Note that by the construction above, if (i) all subblocks of agent $i$ have volume exactly $\delta$ (i.e., there are no imperfect subblocks), (ii) there are no parity beads and (iii) each cut in $\mathcal{S}$ either doesn't intersect any valuation block or is placed on the midpoint $(d_j + d_{j+1})/2$ of some interval $[d_j, d_{j+1}]$ (i.e., at the boundary of one or two valuation subblocks, e.g., see the first cut in Figure 23), then $\mathcal{S}$ is an exact solution to the consensus $1/k$-division problem. However, in addition to the potential existence of imperfect subblocks and the parity beads, the cuts in $\mathcal{S}$ might actually be placed on different points in $[d_j, d_{j+1}]$, because of the presence of beads of different colors which might be placed inside the intervals (e.g., see the second cut in the interval between the last two green (dark) beads in Figure 23 for an example).

Note, however, that such a cut will still lie inside $[d_j, d_{j+1}]$, as otherwise the partition of beads would be imbalanced; therefore, the imbalance in volume for such a cut is at most $\delta$. Since there are at most $(k-1)\ell$ cuts in total, the overall imbalance in volume because of the position of the cuts is at most $(k-1) \cdot \ell \cdot \delta$. Additionally, the imbalance in volume from each imperfect subblock is at most $\delta$, and the overall imbalance in volume because of imperfect subblocks is at most $p_{\mathcal{M}}(n) \cdot \delta$. Finally, the imbalance in volume due to the parity beads is at most $(k-1) \cdot \delta$, since the

parity-preserving procedure can remove at most $k-1$ beads for each agent. In total, the overall imbalance is at most $(k-1)\cdot\ell\cdot\delta + p_{\mathcal{M}}(n)\cdot\delta + (k-1)\cdot\delta$, which is less than $\varepsilon$, by the choice of $\delta$. ∎

**8.2. From Necklace-Splitting to approximate Consensus Division.** In this subsection, we prove that the approximate Consensus-Division solution is at least as hard as Necklace-Splitting; together with the result of the previous subsection, this establishes the computational equivalence of the two problems.

THEOREM 8.4. *The* $(n,\ell,k)$-NECKLACE-SPLITTING *problem is polynomial-time reducible to* $(n,\ell,\varepsilon)$-CON-$1/k$-DIVISION*, when the number of cuts* $\ell$ *is bounded by a polynomial in* $n$ *and* $\varepsilon$ *is inverse-polynomial in* $n$*.*

*Proof.* The idea that we will use for the reduction is very similar to the one presented by Alon [3] for proving that a solution to (discrete) Necklace-Splitting can be obtained from a solution for the continuous version. The proof in [3] starts from an (exact) solution to the continuous problem and proves using induction that it can be transformed into a solution for the discrete version, but appropriately moving some of the cuts, if needed. Here, we explain how to obtain a solution to Necklace-Splitting from an *approximate* solution of the continuous division problem and that this process runs in time polynomial in the number of beads of the necklace.

The main idea is to design an instance of $(n,\ell,\varepsilon)$-CON-$1/k$-DIVISION by representing beads of color $i\in\{1,2,\dots,n\}$ of the instance of $(n,\ell,k)$-NECKLACE-SPLITTING by uniform valuation blocks of agent $i\in\{1,2,\dots,n\}$ that have no overlap between agents. Then, there exists a solution to the Consensus Division problem that does not cut through the intervals and that solution is a valid partition of the necklace. Starting from an arbitrary solution (which might have cuts that intersect the valuation intervals), we will move these cuts (if any) to the endpoints of the intervals one by one, while maintaining the total volume of each portion $O_j$ for $j=1,\dots,k$ unchanged.

More concretely, given an instance $\mathcal{B}$ of $(n,\ell,k)$-NECKLACE-SPLITTING we design an instance $\mathcal{C}$ of $(n,\ell,\varepsilon)$-CON-$1/k$-DIVISION as follows:

- For every color $c_i\in\{1,2,\dots,n\}$ of $\mathcal{B}$, we associate an agent $i$.
- For every bead of color $c_i$, we create a valuation block of width $\delta$ and height $1/\delta$ for some sufficiently small $\delta$, such that the bead lies in the midpoint of the interval corresponding to the valuation block. Note that without loss of generality, we can assume that in $\mathcal{B}$, the beads are sufficiently spread (this does not affect the solution) and therefore there is no overlap between any two valuation blocks, and in fact, the distance between any two valuation blocks is at least $\beta$ for some sufficiently large $\beta$.

Note that by taking $\beta$ to be larger than $2\varepsilon$, we can ensure that in any solution $\mathcal{S}$ of $\mathcal{C}$, each cut intersects with at most one valuation block and therefore all agents have their own designated cuts. In other words, manipulating the positions of the cuts that intersect some valuation interval $[l,r]$ of one agent does not affect the quality of the solution for any other agent, as long as the cuts remain within $[l-\beta/2, r+\beta/2]$ (i.e., they do not move into other valuation blocks).

Now consider a solution $\mathcal{S}$ to $\mathcal{C}$. If for all agents $i\in\{1,2,\dots,n\}$, the cuts do not intersect any valuation blocks, then the solution can be translated verbatim to a solution to $(n,\ell,k)$-NECKLACE-SPLITTING by keeping the cuts at the same positions. However, there might be several cuts that intersect the interior of the valuation intervals; we will refer to those as *bad* cuts. Let $\mathcal{B}_i$ be the set of bad cuts for agent $i$.

We will be able to move these cuts based on the following observation. Consider a bad cut at $e$ intersecting a valuation block of agent $i$, defined on an interval $[l, r]$, and let $j_1$ and $j_2$ be the labels of the pieces on the left side and on the right side of the cut, respectively, (we can assume that $j_1 \neq j_2$; otherwise we can simply remove the cut). Let $o^{j_1} = v_i([l, e])$ and $o^{j_1} = v_i([e, r])$ be the volumes of the subblocks in $[l, r]$ corresponding to each label.

Assume without loss of generality that $o^{j_1} > o^{j_2}$ (the argument of the other case is symmetric). If $o^{j_1} - o^{j_2} \geq \varepsilon$, this implies that for the remaining valuation of agent $i$ (besides $o^{j_1}$ and $o^{j_2}$), there is an excess of volume labelled by $j_2$; otherwise $\mathcal{C}$ would not be an approximate solution to the Consensus Division problem. Then, we can move the cuts accordingly (by also possibly moving some other cuts in the process) such that the two excesses cancel out; we explain how to do that below. Note that since we have started from an approximate solution $\mathcal{C}$ of $(n, \ell, \varepsilon)$-Con-1/$k$-Division, after this procedure, bad cuts might still exist, but they will only account for small discrepancies and can be easily handled; we will refer to those cuts as the *inaccuracy cuts*.

Following [3], we will consider a set of multigraphs (one for each agent) $G_i = (V_i, E_i)$, where $V_i = \{F_1, F_2, \ldots, F_k\}$, i.e., we have one vertex for each one of the $k$ possible labels. Each edge of the graph will correspond to a cut; in particular, there is an edge $(F_a, F_b)$ for each bad cut between two pieces with labels $a$ and $b$, and note that there might be multiple such edges. Note that by the discussion above, if $|v_i(O_a) - v_i(O_b)| \geq \varepsilon$, then the degree of both $F_a$ and $F_b$ is at least 2 and therefore the graph has at least one cycle.

For each agent $i$, we will use two subroutines, one to eliminate all cuts in $B_i$ except for possibly the inaccuracy cuts and the second one to eliminate the remaining bad cuts. For the first subroutine, we will work with the graph $G_i$ and we will eliminate cuts in $B_i$ in steps, by removing edges of the graph, i.e., the graph will be evolving. After each step, the following invariant will be maintained: *The total volume of each partition remains unchanged and the number of bad cuts will be reduced by at least* 1. The subroutine is stated below.

**while** $G_i$ has a cycle **do**

Find a cycle $(F_{j_1}, F_{j_2}, \ldots, F_{j_m})$

**Cycle resolving:** Move all the cuts corresponding to edges on the cycle by the same amount. For the cut corresponding to the first edge of the cycle, move it in the direction that increases[5] the volume of the label $F_{j_1}$ and therefore decreases the volume of the label $F_{j_2}$. We will call such a direction *increasing for* $F_{j_1}$ and *decreasing for* $F_{j_2}$. For any other cut corresponding to an edge $(F_{j_{h-1}}, F_{j_h})$ move it in the direction with the opposite effect of the previous movement with respect to label $F_{j_h}$, i.e., if the previous cut was moved in an increasing direction for $F_{j_h}$, the cut will move in a decreasing direction for $F_{j_h}$. Move the cuts until either of the following occurs:

- Some cut coincides with another cut. In that case, merge all the coinciding cuts and remove the labels of the pieces of volume 0.
- Some cut coincides with the endpoint of an interval.

**end while**

It is clear that at the end of each step of the procedure above, the number of bad cuts is decreased by at least one, either because the cut was merged with another

[5] We can also move the cut in the other direction; that will correspond to the same solution with a permutation of the labels along the cycle.

bad cut or because the cut was moved to the boundary of the interval. Additionally, since all the cuts have been moved by the same amount and for a each vertex in the cycle, one move was in an increasing direction and one was in a decreasing direction, the total volume of each portion $O_j$, for $j = 1, \ldots, k$, remains unchanged. Finally, at the end of the routine, all cycles have been resolved and the graph $G_i$ is acyclic.

Note that since the number of steps is at most $\ell$ and the number of cuts moved in each round is at most $\ell$, the subroutine runs in polynomial time, since the number of cuts $\ell$ is bounded by some polynomial $p_\ell(n)$. If $\mathcal{C}$ was an exact solution, the reduction would be completed here; however, since we start from an approximate solution to the Consensus Division problem, we need a second subroutine to deal with the inaccuracy cuts. The subroutine will essentially transform the approximate solution of $\mathcal{C}$ into an exact solution, which in turn is a solution to $\mathcal{B}$.

This subroutine will be simple: just move each cut to the closest endpoint of the interval $[l, r]$ whose interior it intersects. Note that the imbalance in volume between any two labels $j_1$ and $j_2$ is due to a single bad cut; otherwise the graph $G_i$ would have a cycle. Since each valuation block is constructed to have total volume 1, the cut must lie in $[l, l+\gamma] \cup [r-\gamma, r]$, where $\gamma < \varepsilon \cdot \delta$ and therefore it can unambiguously be moved to the closest endpoint of $[l, r]$. Additionally, this sequence of moves produces an exact solution to $\mathcal{C}$, as otherwise, the original solution would have a discrepancy larger than $\varepsilon$ with respect to at least two partitions $O_{j_1}, O_{j_2}$. Since there are only polynomially many cuts, the second subroutine also runs in polynomial time. This completes the proof. ∎

## 9. Conclusion and further work.

**9.1. Necklace-Splitting with $k$ thieves.** In this paper, we proved the PPA-completeness of the Necklace-Splitting problem when the number of thieves is 2. However, the totality of the problem has been established for any number $k$ of thieves [3]. Could we hope to prove PPA-completeness for the problem for any number of thieves? To this end, we extend our PPA membership result to the case when $k$ is a power of 2 by using the argument of Proposition 3.2 of Alon [3] (here, we take both $k$ and $l$ to be 2). In particular, we will reduce Necklace-Splitting to 4-Necklace-Splitting.

Theorem 9.1. $k$-Necklace-Splitting *is in* PPA*, when* $k$ *is a power of* 2.

*Proof.* We start from an instance of 4-Necklace-Splitting (with four thieves) and we regard it as an instance of Necklace-Splitting (with two thieves), which we solve using an algorithm for the latter problem. The solution is a sequence of intervals defined by the endpoints of the necklace and $n$ cuts, each belonging to one of the two collections (corresponding to the two thieves), such that each collection contains exactly half of the beads of each color. Then, we set the beads that lie in intervals belonging to each collection aside and form two new instances of Necklace-Splitting (essentially by "gluing" the different subintervals of the same collection together); note that each new instance will have an even number of beads of each color, since the initial number of beads from each color was a multiple of 4. Then we run the algorithm again on the resulting instances of Necklace-Splitting to obtain a partition into four collections (two for each individual instance), which consitutes a partition of the 4-Necklace-Splitting into four collections according to the definition of the problem. If $n$ is the number of colors, the total number of cuts is (at most) $3n$, and therefore this partition is a solution to 4-Necklace-Splitting. ∎

The above is a Turing reduction, which can be extended straightforwardly to the case of $k$ a power of 2. We can convert such a reduction into a many-one reduction by applying Theorem 6.1 of [16], which shows that PPA and some related complexity classes are closed under Turing reductions.

What is the computational complexity of $k$-thief Necklace-Splitting, for $k$ not a power of 2? As discussed in [68, 27], the proof that it is a total search problem does *not* seem to boil down to the PPA principle. Right now, we do not even even know if it belongs to PTFNP. Interestingly, Papadimitriou [71] (implicitly) also defined a number of computational complexity classes related to PPA, namely, PPA-$p$, for a parameter $p \geq 2$. PPA-$p$ is defined with respect to an input bipartite graph and a given vertex with degree which is not a multiple of $p$, and the goal is to find another vertex with degree which is not a multiple of $p$ (it follows that PPA=PPA-2). This was done in the context of classifying the computational problem related to Chévalley's theorem from number theory, and it was proven that for prime $p$, Chevally mod $p$ is in PPA-$p$ [71].

It seems very likely that the principle associated with Necklace-Splitting for $k$-thieves is the PPA-$k$ principle. Attesting to this are the recent results of [40], which proved that $k$-Necklace-Splitting is in PPA-$p$, for $p$ which is a prime power. To obtain this result, the authors provided a topological characterization of arguments modulo-$p$ and classified several generalizations of the computational versions of Tucker's lemma and the Borsuk–Ulam theorem as PPA-$p$-complete. The classes PPA-$p$ were also the subject of very recent work that followed the conference papers associated with this paper. In particular, [47] proved that an explicit problem associated with the Chévalley–Warning theorem is complete for PPA-$p$, for any prime $p$. They also presented several relations between these classes for different values of $p$. Similar results of the latter nature, as well as different equivalent definitions of the classes, were independently established by [52].

What about the computational hardness of the problem? What is the hardness of the problem with three thieves? At first glance, it seems like a more complicated problem, but there this is not obvious; for example, there is no way to cause the third thief to be a dummy agent and therefore a straightforward reduction from the case of $k = 2$ is unlikely. However, it is worth mentioning here that the computational equivalence between $\varepsilon$-Consensus-Halving and Necklace-Splitting that we have proved in this paper is actually established between the Necklace-Splitting problem for any $k$ and the corresponding approximate $1/k$-Division problem, a generalization of $\varepsilon$-Consensus-Halving (see [75]); a PPA-$k$-hardness result for $k > 2$ for the latter problem would imply a corresponding result for Necklace-Splitting with $k > 2$.

A first step in this direction was taken by Filos-Ratsikas et al. [39], who showed that the approximate 1/3-Consensus-Division problem is PPAD-hard. This result does not imply any computational hardness for Necklace-Splitting with three thieves though, because it is only obtained for $\varepsilon$ which is inversely exponential, and hence our computational equivalence of section 8 does not apply. Additionally, the proof in [39] does not seem to generalize to inverse-polynomial $\varepsilon$ or a PPA-3-hardness result, and entirely new techniques are likely to be needed to obtain these generalizations.

**9.2. Other open problems.** We have left open the questions of whether $\varepsilon$-Consensus-Halving remains PPA-complete for constant $\varepsilon$ and whether Discrete Ham Sandwich remains PPA-complete when coordinates of points are given in unary. Recall that for the former problem, a PPAD-hardness result is known from [37]; it

would be quite interesting to settle this, to verify whether it is possible for the precision parameter to play such an important role in the problem classification.

In classifying a problem as polynomial-time solvable versus NP-complete, this is usually seen as a statement about its computational (in)tractability. The distinction between PPAD-completeness and PPA-completeness is one of expressive power: we believe that PPAD-complete problems are hard; meanwhile, PPA-complete problems are "at least as hard," but of course are still in NP. The expressive power of totality principles that underpin TFNP problems is a topic of enduring interest [9, 46]; note also the related work on bounded arithmetic discussed by [46]. Our results highlight the distinction between computational (in)tractability and expressive power. In analysing the relationships between these complexity classes, it may be fruitful to focus on expressive power.

Finally, [69] initiates an interesting experimental study of path-following algorithms for 2-thief Necklace-Splitting, obtaining positive results when the number of bead colors is not too large. However, path-following seems to be inapplicable for, say, three thieves. The Necklace-Splitting problem may constitute an interesting class of challenge-instances for SAT-solvers, now that it is known to be a very hard total search problem.

**Acknowledgments.** We thank Alexandros Hollender for detailed and insightful proofreading of earlier versions of this paper, also the reviewers for their careful reading and corrections.

REFERENCES

[1] H. Ackermann, H. Röglin, and B. Vöcking, *On the impact of combinatorial structure on congestion games*, J. ACM, 55 (2008), pp. 25:1–25:22.
[2] J. Aisenberg, M. L. Bonet, and S. Buss, 2-*D Tucker is PPA complete*, J. Comput. System Sci., 108 (2020), pp. 92–103.
[3] N. Alon, *Splitting necklaces*, Adv. Math., 63 (1987), pp. 247–253.
[4] N. Alon, *Some recent combinatorial applications of Borsuk-type theorems*, Algebraic, Extremal and Metric Combinatorics, M. M. Deza, P. Frankl, and I. G. Rosenberg, eds., Cambridge University Press, 1988, pp. 1–12.
[5] N. Alon, *Non-constructive proofs in combinatorics*, in Proceedings of the International Congress of Mathematicians (Kyoto-Japan), Springer, New York, 1990, pp. 1421–1429.
[6] N. Alon and D. B. West, *The Borsuk-Ulam theorem and bisection of necklaces*, Proc. Amer. Math. Soc., 98 (1986), pp. 623–628, https://doi.org/10.2307/2045739.
[7] Y. Babichenko and A. Rubinstein, *Settling the complexity of Nash equilibrium in congestion games*, in Proceedings of the 53rd ACM Symposium on Theory of Computing (STOC), 2021, https://arxiv.org/abs/2012.04327.
[8] E. Batziou, K. A. Hansen, and K. Høgh, *Strong Approximate Consensus Halving and the Borsuk-Ulam Theorem*, preprint, arXiv:2103.04452, 2021.
[9] P. Beame, S. Cook, J. Edmonds, R. Impagliazzo, and T. Pitassi, *The Relative Complexity of NP Search Problems*, J. Comput. System Sci., 57 (1998), pp. 3–19, https://doi.org/10.1006/jcss.1998.1575.
[10] A. Belovs, G. Ivanyos, Y. Qiao, M. Santha, and S. Yang, *On the polynomial parity argument complexity of the combinatorial nullstellensatz*, in Proceedings of the 32nd Computational Complexity Conference (CCC), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
[11] S. N. Bhatt and C. E. Leiserson, *How to assemble tree machines*, in Proceedings of the 14th ACM Symposium on Theory of Computing (STOC), 1982, https://doi.org/10.1145/800070.802179.
[12] N. Bitansky, O. Paneth, and A. Rosen, *On the cryptographic hardness of finding a Nash equilibrium*, in Proceedings of the 56th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, 2015, pp. 1480–1498.
[13] P. V. Blagojević and P. Soberón, *Thieves can make sandwiches*, Bull. Lond. Math. Soc., 50 (2018), pp. 108–123.

[14] P. Bonsma, T. Epping, and W. Hochstättler, *Complexity results on restricted instances of a paint shop problem for words*, Discrete Appl. Math., 154 (2006), pp. 1335–1343, https://doi.org/10.1016/j.dam.2005.05.033.
[15] K. Borsuk, *Drei Sätze über die n-dimensionale euklidische Sphäre*, Fund. Math., 20 (1933), pp. 177–190, https://doi.org/10.4064/fm-20-1-177-190.
[16] S. R. Buss and A. S. Johnson, *Propositional proofs and reductions between NP search problems*, Ann. Pure Appl. Logic, 163 (2012), pp. 1163–1182, https://doi.org/10.1016/j.apal.2012.01.015.
[17] X. Chen, D. Dai, Y. Du, and S. H. Teng, *Settling the Complexity of Arrow-Debreu Equilibria in Markets with Additively Separable Utilities*, in Proceedings of the 50th IEEE Symposium on Foundations of Computer Science (FOCS), 2009, https://doi.org/10.1109/FOCS.2009.29.
[18] X. Chen and X. Deng, *On the complexity of* 2D *discrete fixed point problem*, Theoret. Comput. Sci., 410 (2009), pp. 4448–4456, https://doi.org/10.1016/j.tcs.2009.07.052.
[19] X. Chen, X. Deng, and S. H. Teng, *Settling the complexity of computing two-player Nash equilibria*, J. ACM, 56 (2009), pp. 1–57, https://doi.org/10.1145/1516512.1516516.
[20] X. Chen, D. Durfee, and A. Orfanou, *On the complexity of Nash equilibria in anonymous games*, in Proceedings of the 47th Symposium on Theory of Computing (STOC), ACM, 2015, pp. 381–390.
[21] X. Chen, D. Paparas, and M. Yannakakis, *The complexity of non-monotone markets*, in Proceedings of the 45th Symposium on Theory of Computing (STOC), ACM, 2013, pp. 181–190.
[22] A. R. Choudhuri, P. Hubáček, C. Kamath, K. Pietrzak, A. Rosen, and G. N. Rothblum, *Finding a Nash equilibrium is no easier than breaking Fiat-Shamir*, in Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, 2019, pp. 1103–1114.
[23] B. Codenotti, A. Saberi, K. Varadarajan, and Y. Ye, *The complexity of equilibria: Hardness results for economies via a correspondence with games*, Theoret. Comput. Sci., 408 (2008), pp. 188–198.
[24] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou, *The complexity of computing a Nash equilibrium*, SIAM J. Comput., 39 (2009), pp. 195–259.
[25] C. Daskalakis, C. Tzamos, and M. Zampetakis, *A converse to Banach's fixed point theorem and its CLS-completeness*, in Proceedings of the 50th Symposium on Theory of Computing (STOC), 2018, pp. 44–50.
[26] J. De Loera, X. Goaoc, F. Meunier, and N. Mustafa, *The discrete yet ubiquitous theorems of Carathéodory, Helly, Sperner, Tucker, and Tverberg*, Bull. Amer. Math. Soc., 56 (2019), pp. 415–511.
[27] M. de Longueville and R. T. Živaljević, *The Borsuk–Ulam-property, Tucker-property and constructive proofs in combinatorics*, J. Combin. Theory Ser. A, 113 (2006), pp. 839–850.
[28] A. Deligkas, J. Fearnley, T. Melissourgos, and P. G. Spirakis, *Computing exact solutions of consensus halving and the Borsuk-Ulam theorem*, J. Comput. System Sci., 117 (2021), pp. 75–98.
[29] A. Deligkas, A. Filos-Ratsikas, and A. Hollender, *Two's company, three's a crowd: Consensus-halving for a constant number of agents*, in Proceedings of the 22nd ACM Conference on Economics and Computation (EC), 2021.
[30] X. Deng, J. R. Edmonds, Z. Feng, Z. Liu, Q. Qi, and Z. Xu, *Understanding PPA-completeness*, J. Comput. Syst. Sci., 115 (2021), pp. 146–168.
[31] X. Deng, Z. Feng, and R. Kulkarni, *Octahedral tucker is PPA-complete.*, Electronic Colloquium on Computational Complexity, 24 (2017).
[32] X. Deng, Q. Qi, and A. Saberi, *Algorithmic solutions for envy-free cake cutting*, Oper. Res., 60 (2012), pp. 1461–1476.
[33] H. Edelsbrunner and R. Waupotitsch, *Computing a ham-sandwich cut in two dimensions*, J. Symbolic Comput., 2 (1986), pp. 171–178.
[34] E. Elkind, L. A. Goldberg, and P. Goldberg, *Nash equilibria in graphical games on trees revisited*, in Proceedings of the 7th Conference on Electronic Commerce (EC), ACM, 2006, pp. 100–109.
[35] J. Fearnley, P. W. Goldberg, A. Hollender, and R. Savani, *The complexity of gradient descent: CLS = PPAD ∩ PLS*, in Proceedings of the 53rd Symposium on Theory of Computing (STOC), 2021, https://doi.org/10.1145/3406325.3451052.
[36] J. Fearnley, S. Gordon, R. Mehta, and R. Savani, *Unique end of potential line*, J. Comput. System Sci., 114 (2020), pp. 1–35.
[37] A. Filos-Ratsikas, S. K. S. Frederiksen, P. W. Goldberg, and J. Zhang, *Hardness results for consensus-halving*, in Proceedings of the 43rd International Symposium

on Mathematical Foundations of Computer Science (MFCS), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018, pp. 24:1–24:16.
[38] A. Filos-Ratsikas and P. W. Goldberg, *Consensus halving is PPA-complete*, in Proceedings of the 50th Conference on Theory of Computing (STOC), ACM, 2018, pp. 51–64.
[39] A. Filos-Ratsikas, A. Hollender, K. Sotiraki, and M. Zampetakis, *Consensus-Halving: Does it Ever Get Easier?*, in Proceedings of the 21st Conference on Economics and Computation (EC), 2020, pp. 381–399, https://doi.org/10.1145/3391403.3399527.
[40] A. Filos-Ratsikas, A. Hollender, K. Sotiraki, and M. Zampetakis, *A topological characterization of modulo-p arguments and implications for necklace splitting*, in Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, 2021, pp. 2615–2634.
[41] R. M. Freund and M. J. Todd, *A constructive proof of Tucker's combinatorial lemma*, J. Combin. Theory Ser. A, 30 (1981), pp. 321–325.
[42] S. Garg, O. Pandey, and A. Srinivasan, *On the Exact Cryptographic Hardness of Finding a Nash Equilibrium.*, Cryptology ePrint Archive, Report 2015/1078, 2015.
[43] S. Garg, O. Pandey, and A. Srinivasan, *Revisiting the cryptographic hardness of finding a Nash equilibrium*, in Proceedings of the 36th International Cryptology Conference (CRYPTO), Springer, New York, 2016, pp. 579–604.
[44] C. H. Goldberg and D. B. West, *Bisection of circle colorings*, SIAM J. Algebr. Discrete Meth., 6 (1985), pp. 93–106.
[45] P. W. Goldberg and A. Hollender, *The hairy ball problem is PPAD-complete*, J. Comput. System Sci., 122 (2021), pp. 34–62.
[46] P. W. Goldberg and C. H. Papadimitriou, *Towards a unified complexity theory of total functions*, J. Comput. System Sci., 94 (2018), pp. 167–192.
[47] M. Göös, P. Kamath, K. Sotiraki, and M. Zampetakis, *On the complexity of modulo-q arguments and the Chevalley-Warning theorem*, in Proceedings of the 35th Computational Complexity Conference (CCC), S. Saraf, ed., LIPIcs 169, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020, pp. 19:1–19:42.
[48] F. Grandoni, R. Ravi, M. Singh, and R. Zenklusen, *New approaches to multi-objective optimization*, Math. Programming, 146 (2014), pp. 525–554.
[49] M. Grigni, *A Sperner lemma complete for PPA*, Inform. Process. Lett., 77 (2001), pp. 255–259.
[50] T. P. Hill, *Determining a fair border*, Amer. Math. Monthly, 90 (1983), pp. 438–442.
[51] C. R. Hobby and J. R. Rice, *A moment problem in $L_1$ approximation*, Proc. Amer. Math. Soc., 16 (1965), pp. 665–670.
[52] A. Hollender, *The Classes PPA-k: Existence from Arguments Modulo k*, in Proceedings of the 15th Conference on Web and Internet Economics (WINE), 2019.
[53] P. Hubáček and E. Yogev, *Hardness of continuous local search: Query complexity and cryptographic lower bounds*, SIAM J. Comput., 49 (2020), pp. 1128–1172.
[54] E. Jeřábek, *Integer factoring and modular square roots*, J. Comput. System Sci., 82 (2016), pp. 380–394.
[55] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis, *How easy is local search?*, J. Comput. System Sci., 37 (1988), pp. 79–100.
[56] C. Karthik and A. Saha, *Ham sandwich is equivalent to Borsuk-Ulam.*, in Proceedings of the Symposium on Computational Geometry, 2017, pp. 24–1.
[57] S. Kintali, L. J. Poplawski, R. Rajaraman, R. Sundaram, and S.-H. Teng, *Reducibility among fractional stability problems*, SIAM J. Comput., 42 (2013), pp. 2063–2113.
[58] C. Knauer, H. R. Tiwary, and D. Werner, *On the computational complexity of ham-sandwich cuts, helly sets, and related problems*, in Proceedings of the 28th Symposium on Theoretical Aspects of Computer Science (STACS), 2011, pp. 649–660.
[59] C. Y. Lo, J. Matoušek, and W. Steiger, *Ham-sandwich cuts in $R^d$*, in Proceedings of the 24th Symposium on Theory of Computing (STOC), ACM, 1992, pp. 539–545.
[60] C.-Y. Lo, J. Matoušek, and W. Steiger, *Algorithms for ham-sandwich cuts*, Discrete Comput. Geom., 11 (1994), pp. 433–452.
[61] J. Matoušek, *Geometric range searching*, ACM Comput. Surveys, 26 (1994), pp. 422–461.
[62] J. Matoušek, *Lectures on Discrete Geometry*, Grad. Texts in Math. 108, Springer, New York, 2002.
[63] J. Matousek, *Using the Borsuk-Ulam Theorem: Lectures on Topological Methods in Combinatorics and Geometry*, Springer, New York, 2008.
[64] N. Megiddo, *A note on the complexity of P-matrix LCP and computing an equilibrium*, 1988.
[65] N. Megiddo and C. H. Papadimitriou, *On total functions, existence theorems and computational complexity*, Theoret. Comput. Sci., 81 (1991), pp. 317–324.
[66] R. Mehta, *Constant rank two-player games are ppad-hard*, SIAM J. Comput., 47 (2018), pp. 1858–1887.

[67] F. Meunier, *Discrete splittings of the necklace*, Math. Oper. Res., 33 (2008), pp. 678–688.
[68] F. Meunier, *Simplotopal maps and necklace splitting*, Discrete Math., 323 (2014), pp. 14–26.
[69] F. Meunier and B. Neveu, *Computing solutions of the paintshop–necklace problem*, Comput. Oper. Res., 39 (2012), pp. 2666–2678.
[70] F. Meunier and A. Sebő, *Paintshop, odd cycles and necklace splitting*, Discrete Appl. Math., 157 (2009), pp. 780–793.
[71] C. H. Papadimitriou, *On the complexity of the parity argument and other inefficient proofs of existence*, J. Comput. System Sci., 48 (1994), pp. 498–532.
[72] S. Roy and W. Steiger, *Some combinatorial and algorithmic applications of the Borsuk–Ulam theorem*, Graphs Combin., 23 (2007), pp. 331–341.
[73] A. Rubinstein, *Inapproximability of Nash equilibrium*, SIAM J. Comput., 47 (2018), pp. 917–959.
[74] S. Schuldenzucker, S. Seuken, and S. Battiston, *Finding clearing payments in financial networks with credit default swaps is PPAD-complete*, in Proceedings of the 8th Innovations in Theoretical Computer Science Conference (ITCS), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
[75] F. W. Simmons and F. E. Su, *Consensus-halving via theorems of Borsuk-Ulam and Tucker*, Math. Social Sci., 45 (2003), pp. 15–25.
[76] K. Sotiraki, M. Zampetakis, and G. Zirdelis, *PPP-completeness with connections to cryptography*, in Proceedings of the 59th Symposium on Foundations of Computer Science (FOCS), IEEE, 2018, pp. 148–158.
[77] A. H. Stone and J. W. Tukey, *Generalized "sandwich" theorems*, Duke Math. J., 9 (1942), pp. 356–359.
[78] A. W. Tucker, *Some topological properties of disk and sphere*, in Proceedings of the 1st Canadian Mathematical Congress, Montreal, University of Toronto Press, 1945, pp. 286–309.
[79] V. V. Vazirani and M. Yannakakis, *Market equilibrium under separable, piecewise-linear, concave utilities*, J. ACM, 58 (2011).