The Complexity of Splitting Necklaces and Bisecting Ham Sandwiches

Aris Filos-Ratsikas*[1] and Paul W. Goldberg[2]

[1]Department of Computer Science, École polytechnique fédérale de Lausanne
aris.filosratsikas@epfl.ch
[2]Department of Computer Science, University of Oxford
Paul.Goldberg@cs.ox.ac.uk

November 6, 2018

Abstract

We resolve the computational complexity of two problems known as Necklace-splitting 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. We do this via a PPA-completeness result for an approximate version of the Consensus-halving problem, strengthening our recent result that the problem is PPA-complete for inverse-exponential precision. At the heart of our construction is a smooth embedding of the high-dimensional Möbius strip in the Consensus-halving problem. These results settle the status of PPA as a class that captures the complexity of "natural" problems whose definitions do not incorporate a circuit.

Keywords: Computational complexity; Tucker's Lemma; TFNP; fair division

1 Introduction

The complexity classes PPA and PPAD were introduced in a seminal paper of Papadimitriou [60] in 1994, in an attempt to classify several natural problems in the class TFNP [58]. TFNP is the class of *total search problems* in NP for which a solution exists for every instance, and solutions can be efficiently verified. Various important problems were subsequently proven to be complete for the class PPAD, such as the complexity of many versions of Nash equilibrium [19, 14, 26, 59, 63, 15], market equilibrium computation [18, 12, 71, 16, 65], and others [24, 43]. As evidence of computational hardness, PPA-completeness is stronger than PPAD-completeness, i.e., PPAD $\subseteq$ PPA. Indeed, Jeřábek [40] shows that it indicates cryptographic hardness in a strong sense: [40] gives a randomised reduction from FACTORING to PPA-complete problems. This is not known for PPAD-complete problems. For more details, and the significance of PPA-completeness, we refer the reader to the related discussion in [29]. PPA is the class of problems reducible to Leaf (Definition 1), and a PPA-complete problem is polynomial-time equivalent to Leaf.

*Most of this work was performed when the author was at the University of Oxford.

Definition 1 *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.*

Complete problems for the class PPA seemed to be much more elusive than PPAD-complete ones, especially when one is interested in "natural" problems, where "natural" here has the very specific meaning of problems that do not explicitly contain a circuit in their definition. Besides Papadimitriou [60], other papers asking about the possible existence of natural PPA-complete problems include [36, 13, 19, 22]. In a recent precursor [29] to the present paper we identified the first example of such a problem, namely the approximate Consensus-halving problem, dispelling the suspicion that such problems might not exist. In this paper we build on that result and settle the complexity of two natural and important problems whose complexity status were raised explicitly as open problems in Papadimitriou's paper itself, and in many other papers beginning in the 1980s. Specifically, we prove that Necklace-splitting (with two thieves, see Definition 2) and Discrete Ham Sandwich are both PPA-complete.

Definition 2 (Necklace Splitting) *In the k-*Necklace-splitting *problem there is an open necklace with ka_i beads of colour 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 colour i, $1 \leq i \leq n$.*

In Definition 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 colour are equally shared. In this paper, usually we have $k = 2$ and we refer to this special case as Necklace-splitting.

Definition 3 (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 3, 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.

The *necklace-splitting problem* was introduced in a 1982 paper of Bhatt and Leiserson ([8], Section 5), where it arose in the context of VLSI circuit design (the version defined in [8] 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 2) by Goldberg and West [34] and Alon and West [5] and then in 1987, Alon [2] proved existence of solutions for k thieves as well. Early papers that explicitly raise its complexity-theoretic status as an open problem are Goldberg and West [34] and Alon [3, 4]. 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 [53, 55, 54] explicitly mention the question of the computational complexity of necklace-splitting. Meunier [53] 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 et al. [10], for the "paint shop problem with words", equivalent to 2-thief Necklace-splitting with 2 beads of each colour.)

In [29], we showed Necklace-splitting to be computationally equivalent to ε-Consensus-halving for inverse-polynomial precision parameter ε, but the PPA-completeness of ε-Consensus-halving was only shown for inverse-exponential ε. [29] established PPAD-hardness of Necklace-splitting, applying the main result of [28]. In this paper, we prove that ε-Consensus-halving is

PPA-complete for ε inversely polynomial, thus obtaining the desired PPA-completeness of NECKLACE-SPLITTING. While some structural parts of our reduction are extensions of those presented in [29], obtaining the result for inverse-polynomial precision is much more challenging, as the construction needs to move to a high-dimensional space (rather than the two-dimensional space which is sufficient for the result in [29]). We highlight the main new techniques that we have developed in this paper in Section 2.1, where we provide an overview of the reduction. Our PPA-completeness result gives a convincing negative answer to Meunier and Neveu's questions [54] about possible polynomial-time solvability or membership of PPAD for NECKLACE-SPLITTING; likewise it runs counter to Alon's cautious optimism at ICM 1990 ([4], Section 4) that the problem may be solvable in polynomial time.

The *Ham Sandwich Theorem* [68] is of enduring and widespread interest due to its colourful 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 non-constructive existence proofs and mostly did not touch on the algorithmics. A 1983 paper by Hill [37] 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 [25, 46, 47, 50]. 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 (Definition 3) as named in [60] is essentially this, with d set equal to n to emphasise that we care about the high-dimensional case.) In this work in computational geometry, the emphasis has been on efficient algorithms for small values of d; Lo et al. [47] 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. [35] apply the "Generalized Ham Sandwich Theorem" to a problem in multi-objective optimisation and note that a constructive proof would allow a more efficient algorithm to emerge. The only computational hardness result we know of is Knauer et al. [44] who obtain a $W[1]$-hardness result for a constrained version of the problem; [44] 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. Karpic and Saha [42] 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: Blagojević and Soberón [9] shows a generalisation, 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 Deng et al. [23] (mentioning that both problems "show promise to be complete for PPA"), Aisenberg et al. [1], and Belovs et al. [7].

Further Related Work: The class TFNP was defined in [58] and several of its subclasses were studied over the years, such as PPA, PPAD and PPP [60], PLS [41] and CLS [20]; here we focus on the most recent results. As we mentioned earlier, in [29] we identified the first natural complete problem for PPA, the approximate CONSENSUS-HALVING problem. In a recent paper, Sotiraki et al. [67] 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 et al. [21] and Fearnley et al. [27] identified complete problems (two versions of the Contraction

Map problem, where a metric or a meta-metric are 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 [27] is closely related to the End of Metered Line of [39].

2 Problems and Results

We present and discuss our main results, and in Section 2.1 we give an overview of the proof and new techniques, in particular with respect to the precursors [28, 29] to this paper.

Definition 4 (ε-Consensus Halving [66, 29]) *An instance I_{CH} incorporates, for $1 \leq i \leq n$, a non-negative measure μ_i of a finite line interval $A = [0, x]$, where each μ_i integrates to 1 and $x > 0$ is part of the input. We assume that μ_i are step functions represented in a standard way, in terms of the endpoints of intervals where μ_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 $\varepsilon \geq 0$ also using the bit model. We regard μ_i as the value function held by agent i for subintervals of A.*

A solution consists firstly 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 of A (i.e. A_+ denotes the leftmost label in a Consensus-halving solution).

The Consensus-halving problem of Definition 4 is a computational version of the *Hobby-Rice theorem* [38]. Most of the present paper is devoted to proving the following theorem.

Theorem 2.1 *ε-Consensus-halving is PPA-complete for some inverse-polynomial ε.*

As mentioned in the introduction, in [29] it was proven that 2-thief Necklace-splitting and ε-Consensus-halving for ε inversely-polynomial are computationally equivalent, i.e. they reduce to each other in polynomial time. Therefore, by [29] and the "in PPA" result proven in Section B.1, we immediately get the following corollary.

Theorem 2.2 Necklace-splitting *is PPA-complete when there are $k = 2$ thieves.*

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". Alas, the proofs that Necklace-splitting is a total search problem for $k > 2$ [2, 56] do *not* seem to boil down to the parity argument on an undirected graph! That being said, we do manage to establish membership of PPA for k being a power of 2 (essentially an insight of [2]), see Section B of the Appendix for the details and a related discussion. Of course, the $k = 2$ result strongly suggests that k-Necklace-splitting is a hard problem for other values of k.

As it happens, the PPA-completeness of Discrete Ham Sandwich follows straightforwardly, and we present that next. The basic idea of Theorem 2.3 of embedding the necklace in the moment curve appears already in [62, 51] and [48], p.48.

Theorem 2.3 DISCRETE HAM SANDWICH *is* PPA*-complete.*

Proof. Inclusion in PPA is shown in Section B.1 of the Appendix. For PPA-hardness, we reduce from 2-thief NECKLACE-SPLITTING which is PPA-complete by Theorem 2.2.

The idea is to embed the necklace into the *moment curve* $\gamma = \{(\alpha, \alpha^2, \ldots, \alpha^n) : \alpha \in [0,1]\}$. Assume all beads lie in the unit interval $[0,1]$. A bead having colour $i \in [n]$ 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 γ in at most n points, (e.g. see [51], Lemma 5.4.2), 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). □

A limitation to Theorem 2.3 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 [60], DISCRETE HAM SANDWICH stipulated that each of the n sets of points is of size $2n$, whereas Definition 3 allows polynomial-sized sets. We can straightforwardly extend PPA-completeness to the version of [60] 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.

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 "colours". λ 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 4 that μ_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 Section 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 Overview of the proof

We review the ground covered by the precursors [28, 29] to this paper, then we give an overview of the technical innovations of the present paper.

2.1.1 Ideas from [28, 29]

[28] established PPAD-hardness of CONSENSUS-HALVING. An arithmetic circuit can be encoded by an instance to ε-CONSENSUS-HALVING, by letting each gate have a cut whose location (in a solution) represents the (approximate) value taken at that gate. Agents' valuation functions ensure

that values taken at the gates behave according to the type of gate. A "PPAD circuit" can then be represented using an instance of Consensus-halving.

[29] noted that the search space of solutions to instances as constructed by [28], is oriented. A radical new idea was needed to encode the non-oriented feature of topological spaces representable by PPA. That was achieved by using two cuts to represent the coordinates of a point on a triangular region faving two sides identified to form a Möbius strip. (These cuts are the only ones that lie in a specific subinterval of the interval A of a Consensus-halving instance, called the "coordinate-encoding (c-e) region". The two cuts 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. In order to embed a hard search problem into the surface of a standard 2-dimensional Möbius strip, it was necessary to work at exponentially-fine resolution, which immediately required inverse-exponential ε for instances of ε-Consensus-halving. In [29] we reduced from the PPA-complete problem 2D-Tucker [1] (Definition 5 below), a search problem on an exponential-sized 2-dimensional grid.

In [29], 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 present paper retains this high-level structure (Section 4.1). As in [29] 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; in [29] we used 100 copies. Each copy is called a *circuit-encoder*. The purpose of multiple copies is to make the system robust; a small fraction of copies may be unreliable: as in [29] 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. We re-use the "double negative lemma" of [29] that such a cut is not too harmful. We also adapt a result of [29] that when a cut is moved from the 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.

[29] 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 the above mentioned circuit-encoders, which perform computation on those values. As in [29] we use sensor agents. We obtain a simplification with respect to [29] which is that we do not need the gadgets used there to perform "bit-extraction" (converting the position of a c-e cut into n boolean values). In [29], a solution to an instance of Consensus-halving was associated with a sequence of 100 points in the Möbius-simplex (referred there as the "simplex-domain"), and the "averaging manoeuvre" introduced in [19] was applied. In this paper, for a polynomial $p^C(n)$, we sample a sequence of p^C points in a more elegant manner, again exploiting the inverse-polynomial precision of solutions that we care about.

2.1.2 Technical innovations

As in [29], we reduce from the PPA-complete problem 2D-Tucker [1] (Definition 5). That computational problem uses an exponentially-fine 2D grid, and (unlike [29]), in Section 3 we apply the *snake-embedding* technique invented in [14] (versions of which are used in [22, 23] in the context of PPA) to convert this to a grid of fixed resolution, at the expense of going from 2 to n dimensions. The new problem, Variant high-D Tucker (Definition 7) 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 coloured, that we exploit subsequently. A further variant, New variant high-D Tucker (Definition 8) switches to a "dual" version where a hypercube is divided into cubelets, and points in the hypercube are coloured such that interiors of cubelets are monochromatic. A pair of points is

sought having equal and opposite colours and distant by much less than the size of the cubelets.

We encode a point in n dimensions using a solution to an instance of CONSENSUS-HALVING as follows. Instead of having just 2 cuts in the coordinate-encoding region (as in [29]), 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 [29], the unit 2-simplex). This "Möbius-simplex" (Definition 17; Figure 10) 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.

In Section 5.2 we define a crucially-important coordinate transformation (see Figure 11) 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 un-transformed vectors;
- at the two facets that are identified with each other, the coordinates of corresponding points are the negations of each other; our colouring 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 is the "smooth embedding" referred to in the abstract.

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

- The *twisted tunnel* (Definition 23) is an inverse-polynomially thick strip, connecting the two facets that are identified in forming the Möbius-simplex. It contains at its centre 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 "coloured 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 colours, from $\pm[n]$. (We sometimes refer to these as "colour-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 "blanket-sensor agents") will observe severe imbalances between labels A_+ and A_-, precluding a solution.

In [29], it was relatively straightforward to integrate the subset of the 2-dimensional Möbius-simplex that corresponds with the twisted tunnel, with the parts of the domain where the blanket-sensor agent became active (ruling out a solution) in a way that avoided introducing solutions that fail to encode solutions of TUCKER. In the present paper, that gap has to be "coloured-in" in a carefully-designed way (Section 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 (Sections 6.2, 6.3) become more complicated.

3 Snake embedding reduction

The purpose of this section is to establish the PPA-completeness of New variant high-D Tucker, Definition 8. The snake embedding construction was devised in [14], in order to prove that ε-Nash equilibria are PPAD-complete to find when ε is inverse polynomial; without this trick the result is just obtained for ε being inverse exponential. We do a similar trick here. We will use as a starting-point the PPA-completeness of 2D-Tucker, from [1], which is the following problem:

Definition 5 *(Aisenberg et al. [1]) An instance of 2D-*Tucker *consists of a labelling* $\lambda : [m] \times [m] \to \{\pm 1, \pm 2\}$ *such that for* $1 \leq i, j \leq m$, $\lambda(i, 1) = -\lambda(m-i+1, m)$ *and* $\lambda(1, j) = -\lambda(m, m-j+1)$. *A solution to such an instance 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 5 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 6 *(Aisenberg et al. [1]) An instance of nD-*Tucker *consists of a labelling* $\lambda : [m_1] \times \cdots \times [m_n] \to \{\pm 1, \cdots, \pm n\}$ *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 centre 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 (coordinate-wise) by at most 1.*

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

Definition 7 *An instance of* Variant high-D Tucker *is similar to Definition 6 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 coloured with labels from* $\pm[n]$ *such that all colours are used, and opposite facets have opposite labels, and for* $2 \leq i \leq n$ *it holds that the facet with label* i *(resp.* $-i$*) has no grid-point on that facet with label* i *(resp.* $-i$*).*

Theorem 3.1 Variant high-D Tucker *is* PPA*-complete.*

Informal description of snake embedding A snake-embedding consists of a reduction from kD-Tucker to $(k+1)$D-Tucker, which we describe informally as follows. See Figure 1. Let I be an instance of kD-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 one-third of the unfolded version, the other dimension being unchanged (see the right hand side 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

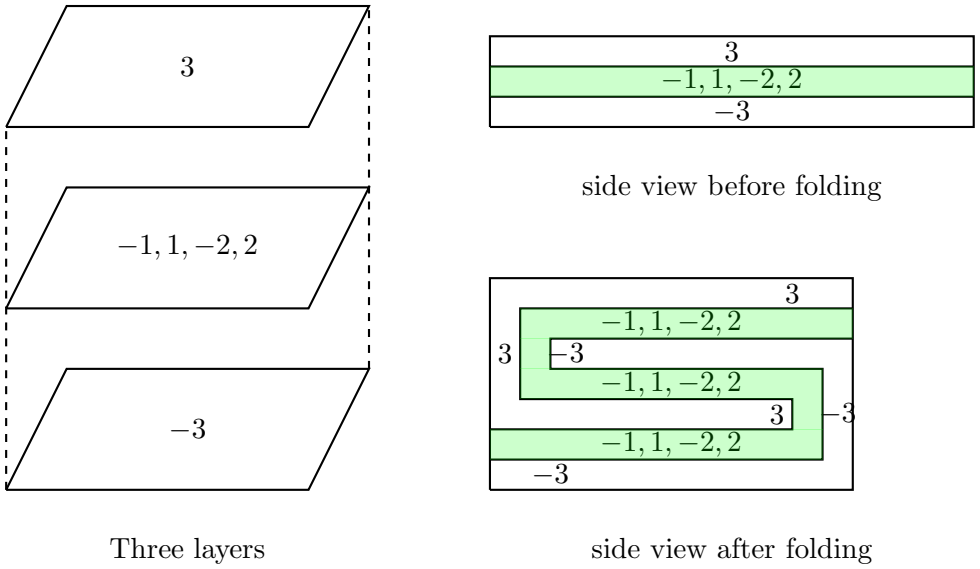


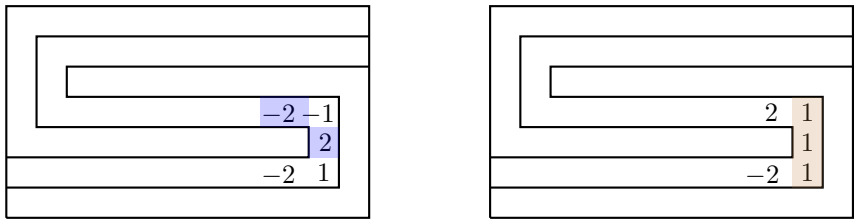
Figure 1: Snake embedding from 2 to 3 dimensions.

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:

- Firstly, 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 colour, whose cross-sections are the short vertical section in the right hand side of Figure 1, see also the right hand side 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.
- Secondly, 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 6, i.e. we might end up with antipodal cubelets that do not have equal-and-opposite colours. To ensure that m_1 is a multiple of 3 before folding, we can add 1 or 2 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.

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



side view without duplication side view with duplication

Figure 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.

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

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 $[m'_1] \times \ldots \times [m_k]$, $\mathbf{x}'$ is mapped to a point $\mathbf{x}$ in $[m_1] \times \ldots \times [m_k]$ and receives a colour $\lambda'(\mathbf{x}')$ such that,

- 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 ℓ 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 $[m_1] \times \ldots \times [m_k]$ with labelling function λ. We will associate each such point with a corresponding point $\mathbf{x}'$ in $\left[\frac{m_1}{3} + 2\right] \times \ldots \times [m_k] \times [7]$ 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 colours (see the shaded cubelets at the right-hand side 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 colours:
  - $\mathbf{x}' = (2, \ldots, x_k, 4)$ *(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 3.1.

Proof of Theorem 3.1. First, it is not hard to check that the the composition of $O(n)$ snake-embeddings is a polynomial-time reduction. Also note that, by the way the high-dimensional instances 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 $2D$ 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 as in [29] (see Figure 1 in that paper), by taking the original $2D$ instance I, of size $m \times m$, and extend to an instance of size $3m \times m$ as follows. The original instance is embedded in the centre 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 2 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 labels. We may assume (by switching 1's and 2's if needed) that these new opposite sides are labelled ± 2.

The S-fold approach shown in Figure 1 (in this paper) can be checked to retain this property. When we sandwich a cuboid between two layers of opposite (new) colours (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 colours of a facet before folding is the same as the set of colours 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 8 *An instance of* New variant high-D Tucker *in n dimensions is presented by a boolean circuit C_{VT} that takes as input coordinates of a point in the hypercube $B = [-1, 1]^n$ and outputs 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 coloured with labels from $\pm[n]$ such that all colours are used, and opposite facets have opposite labels. For $2 \leq i \leq n$ it also holds that the facet with label i (resp. $-i$) does not intersect any cubelet having label i (resp. $-i$). Facets coloured ± 1 are unrestricted (we call them the "panchromatic facets").*

A solution consists of a polynomial number p^C of points that all lie within an inverse polynomial distance $\delta(n)$ of each other (for concreteness, assume $\delta(n) = 1/100n$). At least two of those points should receive equal and opposite labels by C_{VT}.

New variant high-D Tucker corresponds to the problem Variant Tucker in [29]; in that paper a solution only contained 100 points, while here we use p^C points. Here we need more points since we are in n dimensions, and our analysis needs to tolerate n points receiving unreliable labels.

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 encoding of the circuitry is covered in Section 5.

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

- δ^{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 ε (below).
- δ^T is an inverse-polynomial quantity in n, which is smaller than any other inverse-polynomial quantity apart from δ^{tiny} and is larger than δ^{tiny} by an inverse-polynomial amount. The quantity δ^T denotes the width of the so-called "twisted tunnel" (see Definition 23).
- p^{huge} denotes a large polynomial in n; specifically we let $p^{\text{huge}} = n/\delta^{\text{tiny}}$. The quantity p^{huge} represents the number of sensor agents for each circuit encoder (see Definition 13).
- p^{large} denotes a large polynomial in n, which is however smaller than p^{huge} by a polynomial factor. The quantity p^{large} will be used in the definition of the "blanket-sensor agents" (see Definition 14) and will quantify the extent to which the cuts in the "coordinate-encoding region" (Definition 9) are allowed to differ from being evenly spaced, before the blanket-sensor agents become active (see Section 4). The choice of p^{large} controls the value δ_w of the radius of the Significant Region (see Proposition 4.4), with larger p^{large} meaning larger δ_w.
- ε 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 $\varepsilon = \delta^{\text{tiny}}/10$.

4.1 Basic building-blocks

We consider instances I_{CH} of Consensus-halving 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 "localises" 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.

Definition 9 Coordinate-encoding region (c-e region) *Given an 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.*

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 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 4.4.

Definition 10 Coordinate-encoding agents (c-e agents). *Given an instance of* New variant high-D Tucker *in n dimensions, the corresponding instance of* Consensus-halving *has n* coordinate-encoding agents *denoted $\{a_1, \ldots, a_n\}$.*

The n c-e agents have associated n coordinate-encoding cuts (Definition 11). It will be seen that the c-e cuts typically occur in the c-e region. The c-e 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.

Definition 11 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 $p^{\text{huge}} = n/\delta^{\text{tiny}}$ from Section 2, which implies that the c-e region can be divided into p^{huge} intervals of length δ^{tiny} (see also Figure 4).

Definition 12 σ-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* σ-shifted version *of f, when we have that $f'((x - \sigma) \mod n) = f(x)$.*

Recall that the circuit-encoding region (details in Section 5) contains p^C *circuit-encoders*, mentioned in the following definitions.

Definition 13 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*

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

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 non-overlapping intervals of the circuit-encoding region R_i (see Section 5.1). This valuation consists of two valuation blocks of value $\frac{9}{20}$ each, with no other valuation block in between. These are exactly as described in [29], see also Appendix A and Figure 16 for an illustration.

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_-.

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 4). 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 (using the gadget of [29]) 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.

Definition 14 Blanket-sensor agents. *Each circuit-encoder C_i shall have $n-1$* blanket-sensor agents $b_{i,2}, \ldots, b_{i,n}$.

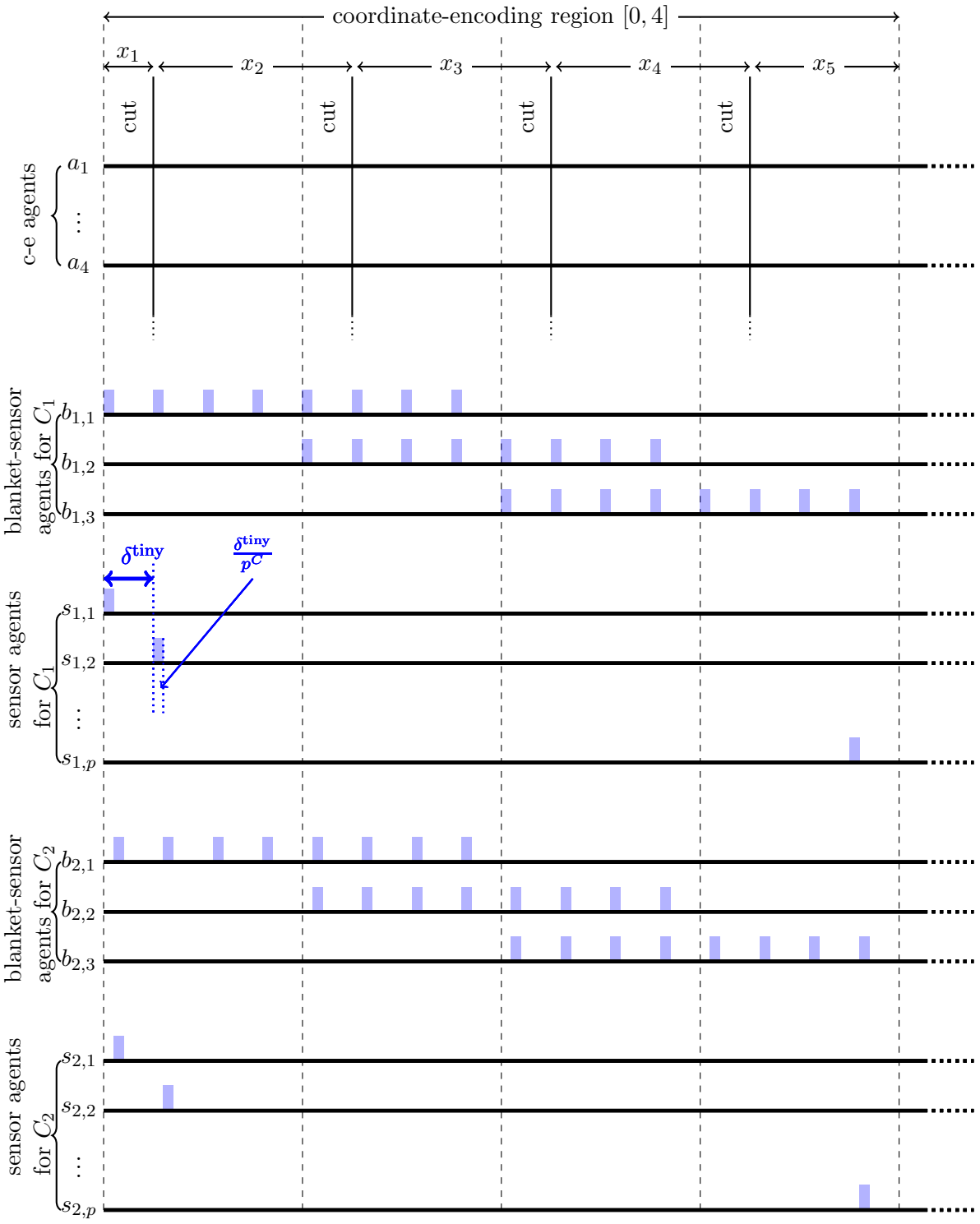


Figure 4: Sensor illustration: example of $n = 4$ c-e cuts representing 5 coordinates summing to 1 (a typical point in the Möbius-simplex). Vertical lines depict the cuts, resulting in labels that alternate between A_+ and A_-, starting with A_+. Shaded blocks over agents' lines indicate value-blocks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

1. *In C_1, for each $j = 2, \ldots, n$, blanket-sensor agent $b_{1,j}$ has value* $1/10$ *distributed over* $[j-2, j]$ *(see Figure 4). This value consists of a sequence*

$$\left[(j-2), (j-2) + \frac{\delta^{\text{tiny}}}{p^C}\right], \cdots, \left[j - \delta^{\text{tiny}}, j - \delta^{\text{tiny}} + \frac{\delta^{\text{tiny}}}{p^C}\right]$$

of $2/\delta^{\text{tiny}} = 2p^{\text{huge}}/n$ *value-blocks, each of length* δ^{tiny}/p^C *and of value* $\frac{1}{10} \cdot (\delta^{\text{tiny}}/2)$.

2. *In each C_i, $1 < i \leq p^C$, 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 3 value-blocks of width* δ^{tiny} *lying in a subinterval $I_{i,j}$ of the circuit-encoding region R_i (see Section 5.1), such that:*

   - *the value-blocks have values*

$$\frac{9(1-\kappa)}{20}, \qquad \frac{9\kappa}{10}, \qquad \frac{9(1-\kappa)}{20}$$

     respectively, where $\kappa = \left(\frac{1}{10}\frac{\delta^{\text{tiny}}}{2}\right)p^{\text{large}}$.

   - *$I_{i,j}$ contains also value-blocks of agents for each gate that takes the value of $b_{i,j}$ as input (the feedback gadgetry, see Section 5.1.2).*

The value of the blanket-sensor agents in $I_{i,j}$ is very similar to the gadget used in [29], see Appendix A (of the present paper) and Figure 17. The structure of the blanket-sensor agents in the c-e region is shown in Figure 4.

Notes on the blanket-sensors

Each blanket-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^{large} (recall that p^{large} is a large polynomial which is however polynomially smaller than p^{huge}) then (in any ε-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; this was referred to as a "bit-detection gadget" in [29], see Appendix A for more details.

Definition 15 (Active blanket-sensor) *We say that blanket-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. We say that $b_{i,j}$ is* active towards A_+ *if A_+ is the overrepresented label, with similar terminology for A_-.*

When blanket-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 over-represented label in $[j-2, j]$ and the parity of the index of the blanket-sensor agent. The precise feedback mechanism to the c-e agent a_j by the blanket-sensor $b_{i,j}$ is described in Section 5.1.3.

When no blanket-sensors are active, the sequence of c-e cuts encodes a point in the Significant Region (Definition 18).

In [29], we worked just in two dimensions and there was just one blanket-sensor agent for the entire c-e region, for each circuit-encoder. Note also that there, the blanket-sensor agent had a single value-block of length 2; here we split it into a polynomial sequence of small value-blocks. The advantage of using a polynomial sequence of value-blocks (which could not have been done in [29] due to the exponential precision requirement) is that we can argue that in all but at most n circuit-encoders, the blanket-sensor agents have value-blocks that are not cut by the c-e cuts, so we can be precise about how big a disparity between blocks labelled A_+ and A_- cause a blanket-sensor to be active, and for at most n circuit-encoders, we regard them as having unreliable inputs (see Definition 16 and Observation 4.2).

4.2 Features of solutions

The main result of this section is Proposition 4.4, 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 4.1 (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 all other cuts are associated with agents who have at least $9/10$ of their value strictly to the right of the c-e region, thus in a solution, those cuts cannot lie in the c-e region.*

Definition 16 (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 4.2 *At most n circuit-encoders fail to receive reliable input (by Observation 4.1 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 c-e agents via value-blocks that get labelled according to the output gates of the circuit being simulated. This is done in a conceptually similar way to that described in [29] (e.g. see Sections 4.4.2 and 4.6 in [29]), see also Appendix A of the present paper.

Definition 17 The Möbius-simplex. *The* Möbius-simplex in n dimensions *consists of points $\mathbf{x}$ in $\mathbb{R}^{n+1}$ whose coordinates are non-negative 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 non-negative $x_1, \ldots, x_n$ 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:*

$$d(\mathbf{x}, \mathbf{x}') = \min\Big(L_1(\mathbf{x}, \mathbf{x}'), \min_{\mathbf{z}, \mathbf{z}' : \mathbf{z} \equiv \mathbf{z}'} (L_1(\mathbf{x}, \mathbf{z}) + L_1(\mathbf{z}', \mathbf{x}'))\Big) \tag{1}$$

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 4.1, 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 17. (Observation 5.3 makes a similar point regarding transformed coordinates.)

Observation 4.3 *Each circuit-encoder reads in "input" representing a point in the Möbius-simplex. 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 (1)). Consequently their collective output (the split between A_+ and A_- of the value held by the c-e agents) is the output of a single circuit-encoder averaged over a collection of p^C points in the Möbius-simplex, all within δ^{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 18 The Significant Region of the Möbius-simplex D. *The* Significant Region *of D consists of all points in D where no blanket-sensors are active (where "blanket-sensors" and "active" are defined in Definition 15).*

Proposition 4.4 *There is an inverse-polynomial value δ^w such that all points $\mathbf{x} = (x_1, \ldots, x_{n+1})$ in the Significant Region have coordinates x_i that for $2 \leq i \leq n$ differ from $1/n$ by at most δ^w, if $\mathbf{x}$ is encoded by the c-e cuts of an ε-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. We start with the following definition.

Definition 19 (Cut δ-close to integer point) *For $\ell \in \{0, \ldots, n\}$, we will say that a cut c is* δ-close to integer point ℓ, *if it lies in $[\ell - \delta, \ell + \delta]$. We will say that cut c is* δ-close to an integer point *if there is some integer $\ell \in \{0, \ldots, n\}$ such that c is δ-close to integer point ℓ.*

Intuitively, cuts that are δ-close to integer points lie close (within distance at most δ) to either the endpoints or the midpoint of some monitored interval $[j-2, j]$.

Note that, by Definition 14, a blanket-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 blanket-sensor agent receive the same label. This will happen if there is a union $\bigcup_\ell I_\ell$ of subintervals I_ℓ of some monitored subinterval $[j-2, j]$, for some $j \in \{2, \ldots, n\}$, which will have total length larger than $1+\delta$, where $\delta > p^{\text{large}} \cdot \delta^{\text{tiny}}$. This means that in $[j-2, j]$, there will be at least $p^{\text{huge}}/n + p^{\text{large}}$ value-blocks of volume $\frac{1}{10} \cdot \frac{\delta^{\text{tiny}}}{2}$ that receive the same label, since by construction, there are at least $p^{\text{huge}}/n + p^{\text{large}}$ such value-blocks in any interval of length at least $1 + p^{\text{large}} \cdot \delta^{\text{tiny}}$. In such a case, the blanket-sensor agent $b_{1,j}$ is active and the set of cuts is not a solution to I_{CH}. In the following, we will consider δ such that $p^{\text{large}} \cdot \delta^{\text{tiny}} < \delta < 1/2n$.

Definition 20 (Monochromatic interval of label A_j) *An interval I is a* monochromatic interval *if it is not intersected by any cuts (which means that it receives a single label). Additionally, 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.

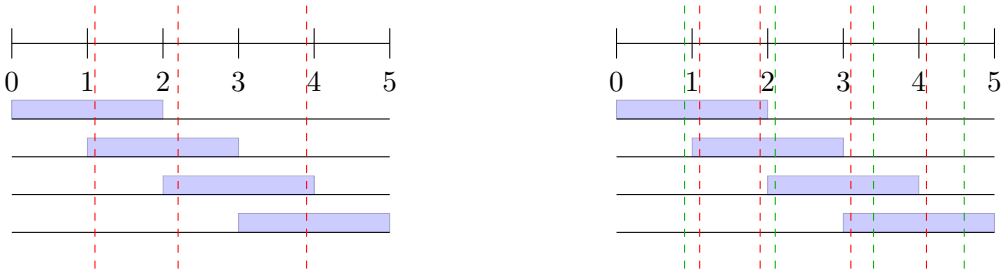


Figure 5: The case of an interval of length k being intersected by $k-2$ (left) or $k-1$ (right) cuts, here $k=5$. The monitored subintervals are depicted in blue. On the left, an interval of length 5 is cut by only 3 cuts. The interval defined by the second and third cuts is of length larger than $1+1/k = 6/5$. On the right, an interval of length 5 is cut by 4 cuts. It is possible to achieve an approximately balanced partition, but only if all cuts are δ-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 blanket-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 3 cuts (right).

It should be clear that if any monitored interval $[j-2,j]$ has a large enough (larger than $1+\delta$) monochromatic subinterval, then the blanket-sensor agent $b_{1,j}$ is active.

Lemma 4.5 *For some $\ell \in \{0, n-k\}$,with $k>1$, consider the interval $I=[\ell,\ell+k]$ of length k and assume that there are at most $k-2$ cuts in this interval. Then at least one of the blanket-sensors monitoring the subintervals in I will be active.*

Proof. In I, there are at least $k-1$ intervals monitored by blanket-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, since the monitored intervals partition the interval I, I_{max} will contain a monochromatic interval of length at least $1+\delta$, which will be entirely contained in some monitored interval, and the corresponding blanket-sensor agent will be active. □

Lemma 4.6 *For some $\ell \in \{0, n-k\}$, with $k>1$, consider the interval $I=[\ell,\ell+k]$ of length k and assume that there are $k-1$ cuts in this interval. Then either*

- *each of the $k-1$ cuts in I will be δ-close to a different integer point and these integer points will be the midpoints of the monitored subintervals contained entirely in I or*
- *at least one of the blanket-sensors monitoring the subintervals in I will be active.*

Proof. In I, there are at least $k-1$ intervals monitored by blanket-sensor agents and we have $k-1$ cuts at our disposal. Assume that there exists some integer point $j-1$ which is a midpoint of some monitored interval $[j-2,j] \subseteq I$ such that there exists no cut that is δ-close to $j-1$. This implies that either $[j-2,j]$ is intersected by at least 2 cuts, or the blanket-sensor agent corresponding to $[j-2,j]$ will be active. To see this, note that if there were not any cuts in $[j-2,j]$ then obviously the whole interval $[j-2,j]$ would be monochromatic and the corresponding blanket-sensor would

be active. If $[j-2, j]$ was intersected by a single cut, since the cut lies at distance at least δ from the midpoint of the interval, there would exist a monochromatic subinterval of $[j-2, j]$ of length at least $1+\delta$, activating the corresponding blanket-sensor agent $b_{1,j}$.

However, for the blanket-sensor agent $b_{1,j}$ to not be active, it would have to be the case that some other cut in the interval $[j-2, j]$ is also not δ-close to the midpoint of one of the adjacent monitored intervals, therefore generating an imbalance in labels that has to be compensated with at least one additional cut in that interval. Given that there are only $k-1$ cuts in the interval I, it follows that in some monitored subinterval $[j', j'-2]$ there will be a large enough imbalance, i.e a monochromatic subinterval of length at least $1+\delta$, and the corresponding blanket-sensor agent $b_{1,j'}$ will be active. See the right-hand side of Figure 5 for an illustration. □

We are now ready to proceed with the proof of Proposition 4.4.

Proof of Proposition 4.4. First, recall that by Observation 4.1, at most n cuts can lie in the c-e region. Also recall that from Definition 14, for the circuit encoder C_1, the blanket-sensor agent $b_{1,j}$, $j \in \{2, \ldots, n\}$ has valuation only in the interval $[j-2, j]$ of the c-e region, i.e. it "monitors" the interval $[j-2, j]$. The blanket-sensor agent $b_{i,j}$ for $i \in \{2, \ldots, p^C\}$ is a $(i-1)\frac{\delta^{\text{tiny}}}{p^C}$-shifted version of $b_{1,j}$. We will make the argument for the blanket-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 blanket-sensor agent will be active.

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 2δ). 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_ℓ, $\ell \in \{1, 2\}$. 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 3 cuts that lie in $[j-2, j]$. We consider three cases.

There are 5 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 4.5, some blanket-sensor agent will be active and we are done.

There are 4 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 4.5, some blanket-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 $[j-2, n]$ (respectively $[0, j-2]$) is intersected by $n-4$ cuts and Lemma 4.5 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 4.6, we know that the cuts in $[j, n]$ are δ-close to integer points and particularly, they are δ-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 blanket-sensor agent to not be activated, which is only possible if one of the 4 cuts in $[j-2, j]$ is δ-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 blanket-sensor agent $b_{1,j}$ would be active. From this discussion, it follows that:

- all the cuts in $[j-2, j]$ are δ-close to integer coordinates within the interval and
- there is at least one cut in $[j-2, j]$ that is δ-close to the midpoint $j-1$ of the monitored interval, one cut that is δ-close to the right endpoint j of the monitored interval and at least one cut that is δ-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 6 for an illustration.

Now, we consider three cases with respect to the positions of the 4 cuts in $[j-2, j]$, illustrated in Figure 7. From the discussion above, we know that three of the cuts will be δ-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.

(A) $\tilde{c}$ is δ-close to $j-1$. In that case, assuming wlog 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 blanket-sensor $b_{1,j}$ will be active. See the left-hand side of Figure 7.

(B) $\tilde{c}$ is δ-close to j. In that case, it is possible that $[j-2, j]$ does not contain a union of monochromatic intervals of the samel label of length at least $1+\delta$. However, by the parity of

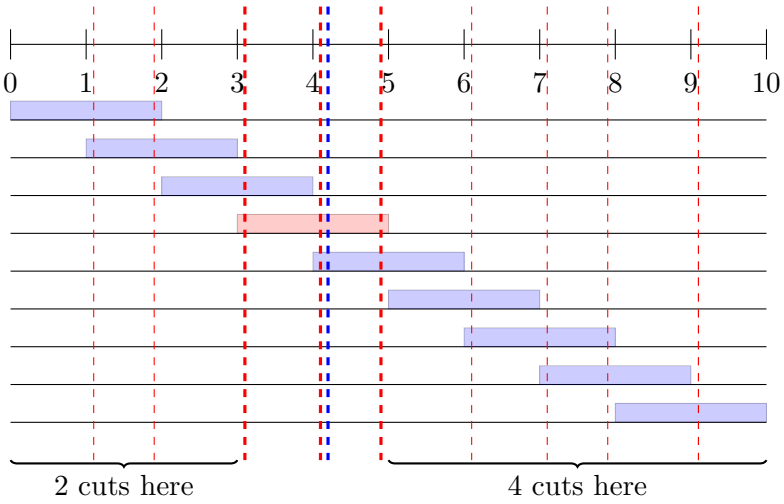


Figure 6: The case in which there are 4 cuts in the interval $[j-2, j]$ (shown in red), here for $j = 5$. The 3 cuts that lie in δ-distance from the left endpoint, midpoint and right endpoint of $[j-2, j]$ are depicted with thick dashed red lines. The other cut in the interval (based on the position of which the different cases are considered) is depicted by a thick dashed blue line and in the particular case, it is shown to be δ-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 δ-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 blanket-sensor agent would be active.

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 blanket-sensor $b_{1,j+1}$. See the right-hand side of Figure 7.

(C) $\tilde{c}$ is δ-close to $j-2$. This case is symmetric to Case (B) above.

There are 3 *cuts in* $[j-2, j]$. Again, considering the intervals $[0, j-2]$ and $[j, n]$ as we did in the case of 4 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 4.5 some blanket-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 4 cuts above, we can establish a very similar statement, namely that:

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

(a) $\tilde{c}$ is δ-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 δ-close to the integer points $j-2, j-1$ and j respectively, both

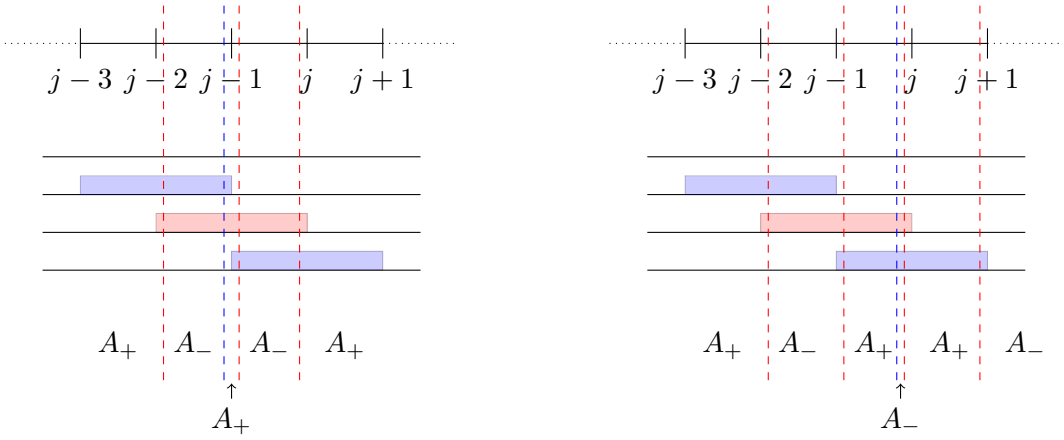


Figure 7: The two subcases of the case when there are 4 cuts in the interval $[j-2, j]$. The three cuts c_1, c_2 and c_3 that are δ-close to the integer points $j-2$, $j-1$ and j in the interval are shown in red, the other cut $\tilde{c}$ is shown in blue. On the left, when $\tilde{c}$ is δ-close to the midpoint $j-1$ of the interval, most of $[j-2, j]$ is coloured with the same label, here A_-, by the parity of the cut sequence. On the right, $\tilde{c}$ is δ-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 δ-close to the right endpoint $j+1$ (shown in red here).

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 8, left-hand side.

(b) $\tilde{c}$ is δ-close to $j-1$. In that case, similarly to Case (A) for the case of 4 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 blanket-sensor $b_{1,j}$. See Figure 8, middle.

(c) $\tilde{c}$ is δ-close to j. Again, similarly to Case (B) for the case of 4 cuts, it is possible that $[j-2, j]$ does not contain a union of monochromatic intervals of the samel 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 blanket-sensor $b_{1,j+1}$. See Figure 8, right-hand side.

This completes the proof. □

Remarks: Looking ahead, certain points in the Significant Region encode NEW VARIANT HIGH-D TUCKER (namely, the ones in the "twisted tunnel", Definition 23). The Significant Region contains the twisted tunnel, being a somewhat wider 1-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 17). Topologically, the Significant Region is a high-dimensional Möbius strip.

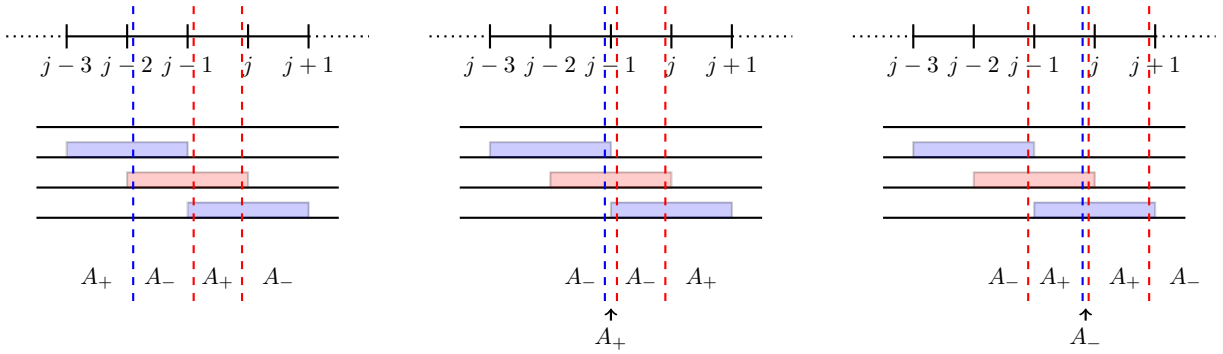


Figure 8: The three subcases of the case when there are 3 cuts in the interval $[j-2, j]$. The two cuts c_1, c_2 and that are δ-close to the integer points $j-1$ and j in the interval are shown in red, the other cut $\tilde{c}$ is shown in blue. On the left, when $\tilde{c}$ is δ-close to the left endpoint $j-2$ 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. In the middle, $\tilde{c}$ is δ-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 on the right, $\tilde{c}$ is δ-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 δ-close to the right endpoint $j+1$ (shown in red here).

5 Reducing from New variant high-D Tucker to Consensus-halving

In Sections 5.1 we give an overview of aspects of how we construct an instance I_{CH} of ε-Consensus-halving (in poly-time) from an instance of New variant high-D Tucker, for inverse polynomial ε. Section 5.2 describes the new coordinate system for the Möbius-simplex D and establishes key properties. Section 5.3 presents a colouring function of D in terms of the coordinate system of Section 5.2. Section 5.4 describes how to construct a purported solution to n-dimensional New variant high-D Tucker from a solution to ε-Consensus-halving. In Section 6 we prove that a solution to New variant high-D Tucker that is obtained by reducing to ε-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 ε-Consensus-halving from an instance of New variant high-D Tucker

We define the reduction from New variant high-D Tucker (Definition 8) to ε-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-encoding agents* $a_1, \ldots, a_n$ (Definition 10). See Figure 9.

- The consensus-halving domain A of I_{CH} has a *coordinate-encoding region* (c-e region) (Definition 9) consisting of the interval $[0, n]$.
- I_{CH} has p^C *circuit-encoders* (Sections 5.1.1, 5.1.4), $C_1, \ldots, C_{p^C}$.
  - Each C_i has a set $\mathcal{A}_i$ of agents (see Figure 9) 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.
  - Each C_i has p^{huge} *sensor agents* as defined in Definition 13 each of which has a block of value $1/10$ in a small subinterval of the c-e region as specified in Definition 13, and further value in region R_i.
  - Each C_i has $n-1$ *blanket-sensor agents* as in Definition 14.

Remarks: We associate one cut with each agent; let $c(a)$ be the cut associated with agent a. The cuts $c(a_i)$ for coordinate-encoding agents, are called the *coordinate-encoding cuts* (or c-e cuts). A straightforward consequence of Proposition 4.4 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, similarly to [29] we have that the duplication of the circuit using p^C circuit-encoders, allows the circuitry to be robust to this problem. See Appendix A for more details.

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 coloured 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 constructed in [29] (see Appendix A for a more detailed exposition). $a(g)$'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 18). The positions of these cuts affect the labels of blocks of value held by the n coordinate-encoding agents, as detailed in Section 5.1.2.

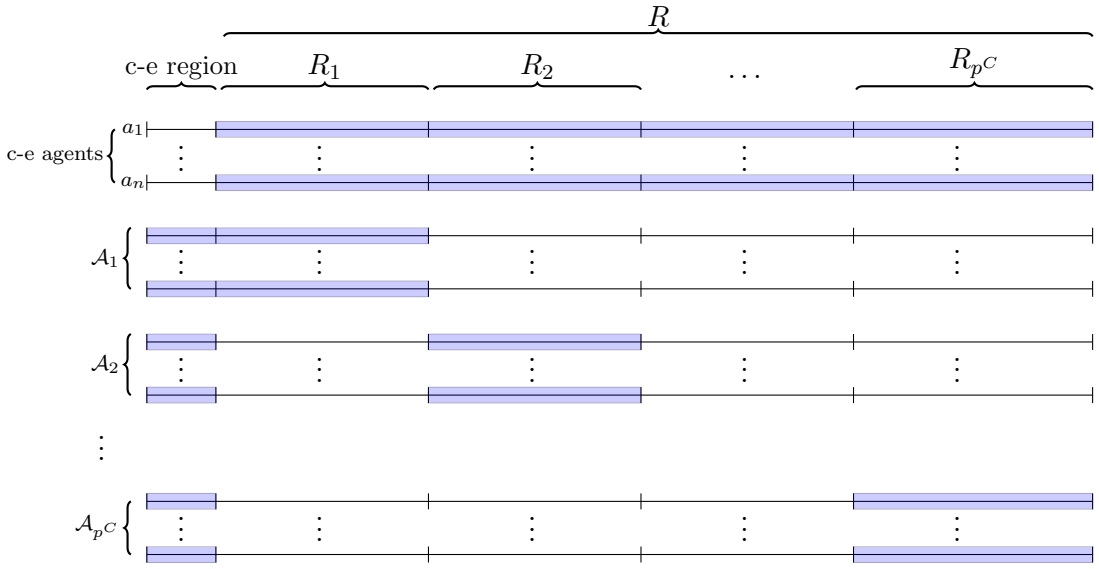


Figure 9: An overview of I_{CH}, denoting all the different regions and the agents of $C_1 \ldots, C_n$, as well as the coordinate-encoding agents. The highlighted areas denote that the corresponding agent has non-zero value on these regions.

Definition 21 Reference sensor-agent. *Noting from Definition 13 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*[1] *$s_{1,1}$, in the sense that after simulating C_{VT} we take the exclusive-or with $s_{1,1}$.*

We used this crucial technique of Definition 21 in [29]: it performs the task of disorienting the domain while at the same time ensuring continuity when we move a cut from the left-hand side of the c-e region to the right-hand side.

Preprocessing, prior to simulating C_{VT} For each C_i, we take all p^{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 17). As noted earlier (Observation 4.2) 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.

We then perform a coordinate transformation described in Section 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 8). These points get their coordinates passed directly to a copy of C_{VT}, and the outputs of C_{CV} are used to provide feedback to the c-e agents as described in Section 5.1.2 (and discussed in Observations 4.2 and 4.3). Other points get coloured in a manner that avoids allowing bogus solutions to I_{CH} (i.e. ones that do not encode solutions to I_{VT}).

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

5.1.2 Output gates of C_1, and the feedback they provide to the coordinate-encoding agents

The following generalises [29]. 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 [28, 29] (see Appendix A). Let $x_{REF} \in \{\text{TRUE}, \text{FALSE}\}$ be the negation of the value of the reference sensor (Definition 21). 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-e agents $a_1, \ldots, a_n$ has 2 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 20 in Appendix A shows an example of how this feedback works.

5.1.3 How C_1's blanket-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 blanket-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 Section 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-e 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-e 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-e agent as feedback, as the circuit can convert the input value encoded by the value gadget of the blanket-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 blanket-sensor agent is active, they all affect their corresponding gates. The reason for requiring blanket-sensors of different parities to feedback different labels to the c-e agents is to be consistent with the definition of "consistent colours", see Definition 24.

Note that we do not define the behaviour of the blanket-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-e agent more of the over-represented or under-represented label, depending on their parity, which can be done using gadgetry of [29].

Comment. Consider the operation of moving a cut from near the left-hand side of the c-e region to the right-hand side, 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 17. Suppose also that within the c-e region, we do not change the label of any point. Then the blanket-sensor agents behave the same way: if some blanket-sensor agent sees an excess of A_+ in its interval then it will continue to see an excess of A_+. Regarding the (non-blanket) sensor agents, our reduction will make them "want" to produce opposite outputs, but due to the flipping of x_{REF}, the reference sensor value, 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; see [29], Section 4.4.3 for the precise definition.
- For all i, j, all x in the c-e region, $\mu_{a_{i,j}}(x)$, is specified in Definition 14.

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-e 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 5.1 *The value that is labelled A_+ held by any c-e 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 ℓ, then a_ℓ 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 non-negative 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 10. For $\tau \in [0,1]$, let $\mathbf{x}_\tau$ be the point

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

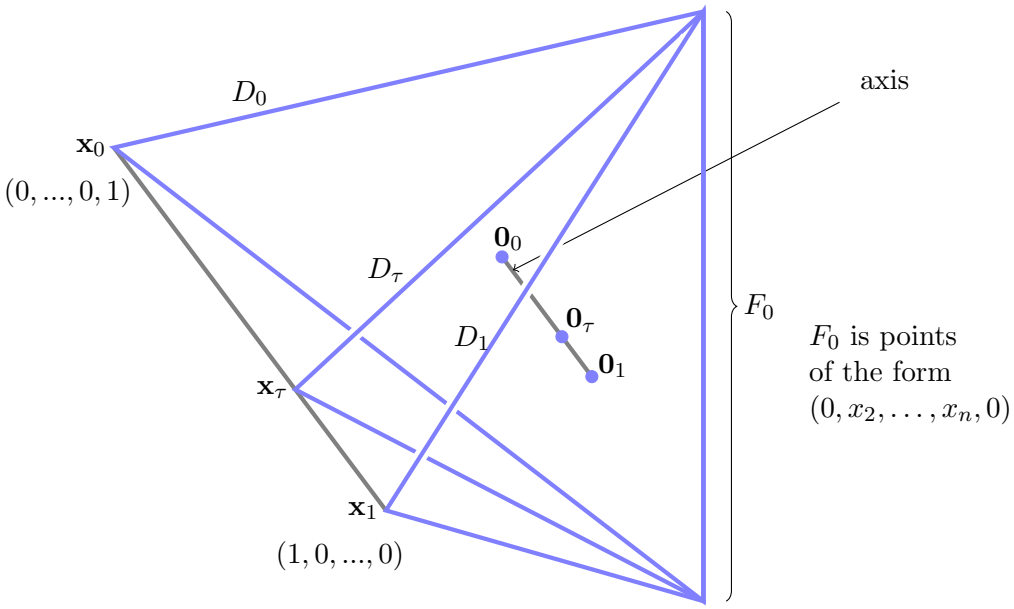


Figure 10: Subspaces of the Möbius-simplex D: D_0 is the triangle spanned by $\mathbf{x}_0$ and F_0, and $\mathbf{0}_0$ is its centre; similarly for D_τ and D_1.

(So, $\mathbf{x}_0$ and $\mathbf{x}_1$ are the endpoints of the 1-dimensional edge of D that is not contained in F_0.) Let D_τ 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 17.

D_τ contains the point $\mathbf{0}_\tau = (\tau/n, 1/n, \ldots, 1/n, (1-\tau)/n)$, which we regard as the origin of D_τ. 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 will be in the Significant Region).

We then refer to points in D_τ by means of the coordinates in a coordinate system that itself is a linear function of τ. With respect to any fixed $\tau \in [0,1]$ we define $n-1$ 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_τ (so that with respect to the origin $\mathbf{0}_\tau$, any point in D_τ has unique coordinates). The other key feature (Observation 5.3) 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^τ:

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

d_2^τ consists of increasing the second coordinate at the expense of its neighbours. For small τ we increase mainly at the expense of the third coordinate and as τ 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 $2 \leq i \leq n$ we define

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

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

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

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

$$d_{-i}^{\tau} := -d_i^{\tau}. \tag{3}$$

Observation 5.2 makes the important point that by linearity, the vectors d_i^{τ}, $i = 2, \ldots, n$, can be used as a coordinate system to refer to points in D_τ.

Observation 5.2 *Any point* $\mathbf{x}$ *in* D_τ *can be uniquely expressed as a sum*

$$\mathbf{x} = \mathbf{0}_\tau + \sum_{i=2}^{n} \alpha_i d_i^{\tau}. \tag{4}$$

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* τ*, note that the vectors* d_i^{τ} *are linearly independent, and the reason why they span* D_τ *is that any vector* d_i^{τ} *is equal to* $(1-\tau)$ *multiplied by a vector in* D_0*, added to* τ *multiplied by an equal-length vector in* D_1*. So these vectors do indeed lie in* D_τ*.*

Definition 22 *For a point* $\mathbf{x} \in D_\tau$ *as in (4) 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* τ *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 (1),* L_1 *denotes the standard* L_1 *distance on vectors.*

$$\tilde{d}(\mathbf{x}, \mathbf{x}') = \min\Big(L_1(\mathbf{x}, \mathbf{x}'), \min_{\mathbf{z},\mathbf{z}':\mathbf{z}\equiv\mathbf{z}'} (L_1(\mathbf{x}, \mathbf{z}) + L_1(\mathbf{z}', \mathbf{x}'))\Big) \tag{5}$$

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

Observation 5.3 *With regard to Definition 22, consider two points* $\mathbf{x} = (0; \alpha_2, \ldots, \alpha_n)$*,* $\mathbf{x}' = (1; -\alpha_2, \ldots, -\alpha_n)$*, 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 11) when we move from a point in* $D_{1-\varepsilon}$ *to a nearby point in* D_ε*, for any* $j \in \{2, \ldots, n\}$*, the direction of increasing* α_j *segues* smoothly *to the direction of* decreasing α_j*.*

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

Proposition 5.4 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 5.4 (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 (1),(5), 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, it follows that there are also inverse-polynomial lower bounds on the resulting perturbations of the output values.

The identification of transformed coordinates $(0; \alpha_2, \ldots, \alpha_n)$ and $(1; -\alpha_2, \ldots, -\alpha_n)$ is of course equivalent to the identification of untransformed coordinates $(0, x_1, \ldots, x_n)$ and $(x_1, \ldots, x_n, 0)$ in (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_∞, since these are polynomially related to L_1.

Note that Proposition 5.4 does not hold for all points in D; the restriction to a neighbourhood of the axis is needed. For points on F_0, all values of τ are equivalent, and for points close to F_0, perturbed versions of them could result in large perturbations of τ.

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 τ for the transformed coordinate system: Note that $(x_1, \ldots, x_{n+1})$ must be a convex combination of x_τ 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 τ can be computed as the solution to the equation

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

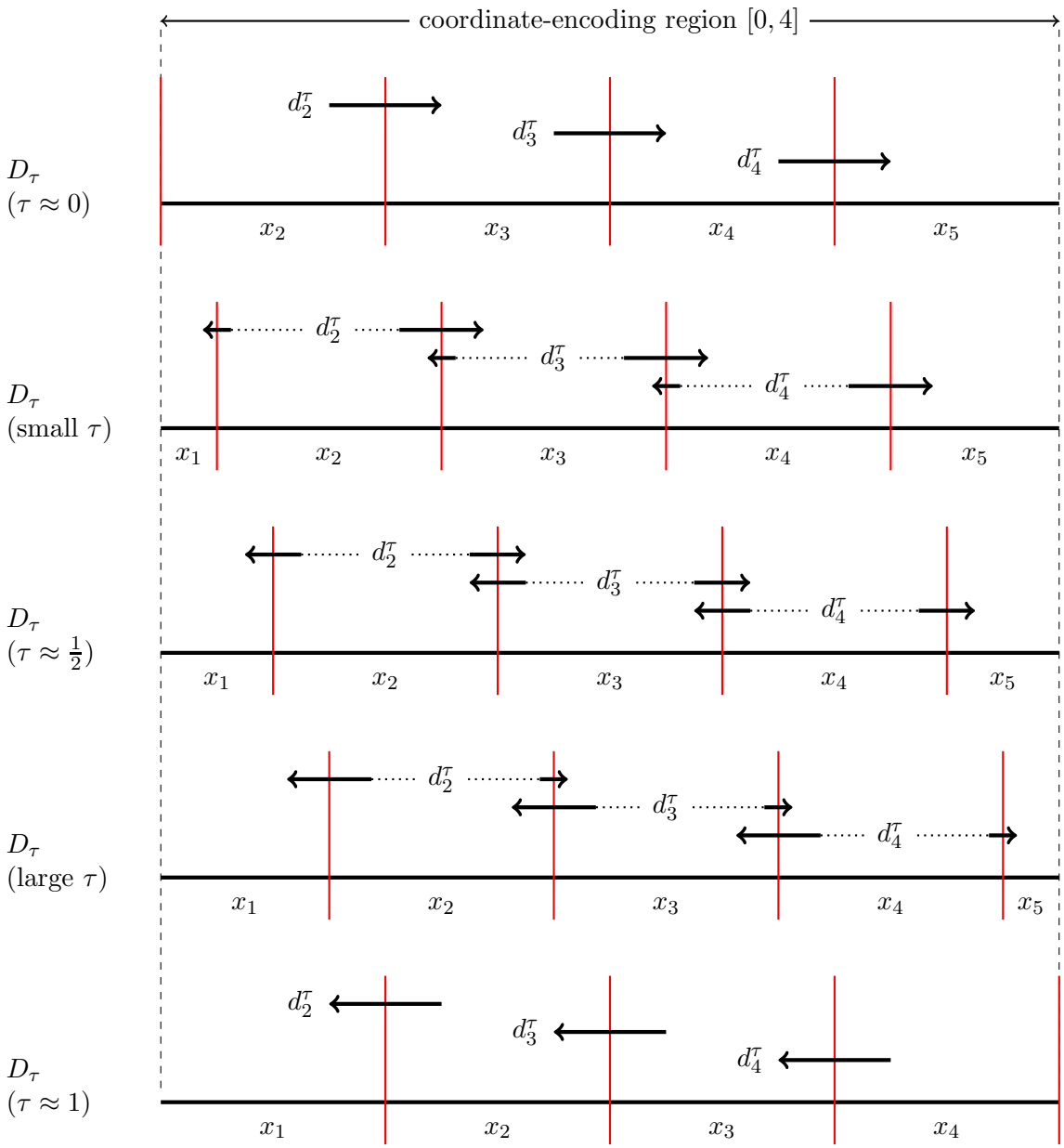


Figure 11: The diagram shows (for $n = 4$) sets of cuts (in red) that correspond to points on the axis, for various values of τ. It also shows how movements of the cuts correspond to movement of a point in D away from the axis. For example, for small τ, a move in direction d_2^τ corresponds to moving the second cut to the right and the first only slightly to the left. Generally, a movement in direction d_i^τ tends to increase x_i at the expense of x_i's neighbours x_{i-1} and x_{i+1}. As τ increases, the movement in direction d_i^τ 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 τ approaches 0 from above, the direction d_i^τ approaches the *negative* of the limit approached by d_i^τ when τ approaches 1 from below.

Note that the dependence of τ 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 α_2, α_3 and so on successively, using the derived formulas for α_i for the computation of α_{i+1} and express each α_i as a function of only τ 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 α_i values might "disappear"; for example, for $\tau = 0$, expressing α_3 in terms of only τ, x_1 and x_2 is not possible, since for $\tau = 0$ we do not obtain a formula for α_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 polynomial-size perturbations of the transformed coordinates. 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 α_i get negated.

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

5.3 A (poly-time computable) partial colouring function $f : D \to \{-1, 0, 1\}^n$

This section defines a partial function $f : D \to \{-1, 0, 1\}^n$ (D being the n-dimensional Möbius-simplex (Definition 17)). 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 18) which is the set of points where no blanket-sensor agents (Definition 14) are active, thus it includes the twisted tunnel T (Definition 23). f is computable by a circuit C, that 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 colour j, $e_f(\mathbf{x})_j = -1$ means that the point receives colour $-j$ and $e_f(\mathbf{x})_j = 0$ means that the point does not receives colour j or $-j$. In general, $e_f(\mathbf{x})$ may

have multiple non-zero entries. We will use the term *the colour of* $\mathbf{x}$ for points that only receive a single colour (and therefore their outputs are vectors with only one non-zero entry).

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

In Section 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 non-negative and sum to 1. And, the "Significant Region" (Definition 18) 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 4.4). (δ^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 coloured in Definition 8. 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 8), and for $i \geq 2$ the facet of B with maximum x_i ($x_i = 1$) consists of points that do not have colour i, and the the facet of B with minimum x_i ($x_i = -1$) consists of points that do not have colour $-i$.

Definition 23 *The* twisted tunnel T *is defined as follows. The* axis *of T is the set of all points $\mathbf{0}_\tau$ as defined in Section 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 δ^T is an inverse polynomial quantity sufficiently small that T is a subset of the Significant Region; this is achieved since by definition, δ^T is polynomially smaller than δ^w of Proposition 4.4. Thus, T has (with respect to the transformed coordinates) a $(n-1)$-cube-shaped intersection with any D_τ.*

We define the behaviour of f over the Significant Region (Definition 18) in 3 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_τ, where we choose $\tau = \frac{1}{2} + \delta^T \cdot x_1$. Then (noting (4)) 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 colour; the colour of $g(\mathbf{x})$ — i.e. the non-zero entry of $f(g(\mathbf{x}))$ — is set equal to the colour allocated to $\mathbf{x}$ in B by I_{VT}. (Notice that the centre 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 centre of the Significant Region. This point has (recalling Definition 22) transformed coordinates $(\frac{1}{2}; 0, \ldots, 0)$ where the first entry is the value of τ.)

2. **Extending f to be defined on T**

   We also colour other points in T as follows — these will also receive single colours. Suppose $\mathbf{y}$ belongs to D_τ, where $\tau < \frac{1}{2} - \delta^T$ or $\tau > \frac{1}{2} + \delta^T$. According to (4), $\mathbf{y} = \mathbf{0}_\tau + \sum_{i=2}^{n} \alpha_i d_i^\tau$, and $\mathbf{y}$ has transformed coordinates $(\tau; \alpha_2, \ldots, \alpha_n)$. Suppose all the α_i lie in the range $[-\delta^T, \delta^T]$. Then if $\tau < \frac{1}{2} - \delta^T$, we set the colour of $\mathbf{y}$ to the colour 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 22) are the same for $\mathbf{y}$ and for $\mathbf{y}'$. We do a similar thing for points in D_τ for $\tau > \frac{1}{2} + \delta^T$. That is, if $\mathbf{y}$ has

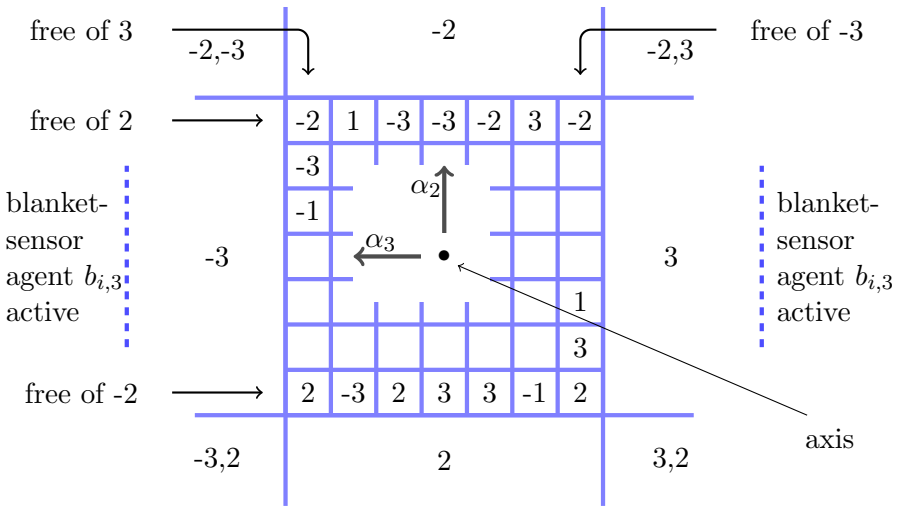


Figure 12: Cross-section of the twisted tunnel (for $n = 3$), with examples of possible labels of regions. Note that the outer regions of the Significant Region are not adjacent to any cubelet having an opposite colour to that outer region. For example, the left-hand column is free of cubelets with colour 3, and is adjacent to the outer region with colour -3.

transformed coordinates $(\tau; \alpha_2, \ldots, \alpha_n)$ where $\tau > \frac{1}{2} + \delta^T$ and the α_i are all at most δ^T in absolute value, then $\mathbf{y}$ gets the same colour 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 18) is points in D where no blanket-sensor agents are active, a subset of points that are close to the axis in the sense of Proposition 4.4. 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 colour $-j$;

   (b) For each $j \in \{2, \ldots, n\}$ if $\alpha_j < -\delta^T$ then $\mathbf{x}$ gets colour j;

   (c) these are not mutually exclusive, $\mathbf{x}$ gets at least one colour, possibly more.

   Notice that (within the subspace D_τ) the side(s) of the twisted tunnel T closest to $\mathbf{x}$ is guaranteed not to be opposite to any colour of $\mathbf{x}$.

   For a subset S of colours, let $R(S)$ be the region with colours in S. We call these the "outer regions".

Proposition 6.7 notes that when colour-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 4.4 already tells us that $\mathbf{x}$ must lie within some inverse-polynomial distance of the axis, since if not,

some blanket-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 colour-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 (Section 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 6.4 tells us that this is always possible.

Now we have two points $\mathbf{x}'$ and $\mathbf{x}''$ within distance δ^{tiny} of each other, that lie in oppositely coloured 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 5.4 to be much smaller than the widths of the cubelets and other colour-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 colour-regions that are adjacent and with opposite colours. Since these colour-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 Completing the Proof of Theorem 2.1

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 ε-approximate solution S_{CH} to I_{CH} allows a solution to I_{VT} to be recovered using Section 5.4.

In S_{CH}, only the c-e cuts may lie in the c-e region (Observation 4.1), and every other cut $c(a)$ for a not a c-e 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 4.4 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, in which case it becomes what we called a "stray cut" in [29], and in that case, the "double negative lemma" of [29] may be applied to prove that it has little effect on the quality of a solution: it causes a single circuit-encoder to have unreliable output.

Recalling Observation 4.3, 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 (1)). In transformed coordinates we have $\tilde{d}(\mathbf{x}_i, \mathbf{x}_j) \leq \tilde{\delta}^{\text{tiny}}$ ($\tilde{d}$ as in (5)), and by Proposition 5.4, $\tilde{\delta}^{\text{tiny}}$ 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, up to $n-1$ circuit-encoders receive unreliable input and one circuit-encoder is affected by the stray cut. We proceed by case analysis. Note that there is an inverse-polynomial gap between the twisted tunnel and the boundary of the Significant Region, which is much larger than δ^{tiny}. So the cases to consider are:

- Most or all of the points are in the twisted tunnel. In that case it will be proved that the procedure of Section 5.4 identifies a solution to NEW VARIANT HIGH-D TUCKER; see Sections 6.1, 6.2.

- Most or all are in the outer regions. In that case we are not at a solution since the colours

cannot cancel each other.

- Some of the points are outside the Significant Region. In that case we are far from the twisted tunnel, and Section 6.3 argues that no solution is possible here.

- All points are outside the Significant Region. Then, we are certainly far from a solution since various blanket-sensor agents will be active, and the points are so close together (within δ^{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 17). We start by defining a function $f' : D \to [-1, 1]^n$ based on f defined as in Section 5.3. f' simulates the effect of the blanket-sensor agents as described in Section 5.1.3. Let $\mathbf{x} \in D$.

1. **When no blanket-sensor agents are active at x.** Here we are in the Significant Region.
   - In T, f' behaves like f in the sense that if f assigns the colour i to $\mathbf{x}$ (recall that it assigns a single colour 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 blanket-sensor agents are active.** If the j-th blanket-sensor agent of C_1, $b_{1,j}$ is active towards 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 blanket-sensor agents, thus $f'(\mathbf{x})$ can contain multiple 1's and -1's.

The following points are similar to Observation 4.3:

Observation 6.1 *Suppose that circuit-encoder C_i (some $i \in [p^C]$) of I_{CH} receives reliable inputs. (Observation 4.2 tells us that at most n of them fail to receive reliable inputs.) Then C_i computes f' at a point within distance δ^{tiny} from the $\mathbf{x} \in D$ encoded by the c-e cuts, in the sense that the value observed by each c-e 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 Section 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 6.2 provides the details.

Proposition 6.2 *I_{CH} computes a function F in the following sense. Let $\mathbf{x}$ be the point encoded by the c-e agents. Suppose all agents other than the c-e agents have error (i.e. discrepancy between A_+ and A_- that they observe) at most ε. Then the error of the c-e agents is within additive distance $1/n$ from the average value of f', averaged over a set of points all within δ^{tiny} of $\mathbf{x}$.*

Proof. We put together various observations about the way I_{CH} is constructed. Observation 4.3 told us that the values observed by the c-e agents are the average of a set of points all within distance δ^{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 4.2), 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 $2n^2$. □

Proposition 6.3 *δ^{tiny} can be chosen to be sufficiently small (but still inverse-polynomial) that given a set of p^C points $\mathbf{x}_1, \ldots, \mathbf{x}_{p^C} \in D$ within distance δ^{tiny} of each other, when we compute their transformed coordinates $\mathbf{y}_1, \ldots, \mathbf{y}_{p^C}$, we have:*

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 colour-region, or adjacent colour-regions (where a "colour-region" is one of monochromatic regions of Section 5.3), or one of the outer regions.

Proof. In identifying which colour-region a point with transformed coordinates $\mathbf{y}$ belongs to, for the colour-regions in the twisted tunnel T, we compare coordinates with certain threshold values. These threshold values differ from each other by inverse-polynomial amounts, and the smallest difference between any pair of them is inverse-polynomial. Applying Proposition 5.4, we can keep the $\mathbf{y}_i$ closer to each other than this. (Colour-regions lie in the Significant Region, so these points lie within $1/10n^2$ of the axis, so Proposition 5.4 is applicable.)

This applies also to the outer regions $R(S)$ for sets of colours S. Identifying which $R(S)$ a point $\mathbf{y}$ belongs to, uses the same information on comparisons of its coordinates with inverse-polynomials. □

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_∞ 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 6.4 *Let S_{CH} be an ε-approximate solution to I_{CH}. Suppose that the c-e cuts of S_{CH} represent a point $\mathbf{x}$ that lies in the twisted tunnel. Then we can reconstruct a solution to I_{VT} in polynomial time.*

Recall that S_{CH}, I_{CH} and I_{VT} and ε are as introduced at the start of Section 6.

Proof. Observation 6.1 tells us that if a circuit-encoder C_i receives reliable inputs, it outputs the colour of a point in the Möbius-simplex that lies within δ^{tiny} of $\mathbf{x}$.

We note next that the feedback received by the c-e 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 $B = [-1, 1]^n$ is obtained by taking c-e 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 centre (or origin) of B iff the c-e 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 centre of B iff the c-e agents have approximately balanced allocations of A_+ and A_-.

Observation 4.2 tells us that at most n circuit-encoders fail to receive reliable input. If all circuit-encoders received reliable input, then the total error at a solution would be at most ε, i.e. the precision parameter of the I_{CH} instance. However, since at most n of them receive unreliable input, we might have an added discrepancy of at most n/p^C when taking the average and therefore we need to get within distance $\varepsilon + \frac{2n}{p^C}$ of the centre of B. For this to be possible, we need some of the cancelling to take place amongst the outputs of the circuit-encoders that received reliable inputs, so we really can find a pair of correctly oppositely-coloured points.

Proposition 6.3 tell us that if $\mathbf{x} \in D$ is the point in D represented by some Consensus-halving solution, then provided $\mathbf{x}$ lies in the twisted tunnel, the corresponding cluster of p^C points must be mapped by the circuit-encoders C_i that mostly cancel each other out, so we find pairs of points that belong to oppositely-labelled colour-regions, from which Section 5.4 tells us how to recover two oppositely-coloured cubelets of I_{VT}. □

It remains to rule out the possibility of $\mathbf{x}$ occurring outside the twisted tunnel.

6.3 No bogus approximate-zeroes of F at boundary of Significant Region

Proposition 6.4 tells us that ε-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 colours. But we have to rule out points with colour $j \in S$ being averaged with nearby points that are "coloured" $-j$ due to a blanket-sensor agent. In more detail, if $\mathbf{x} \in R(S)$ and $\mathbf{x}$ is within δ^{tiny} of $\mathbf{x}'$ for which the j-th blanket-sensor agent is active and provides feedback corresponding to $-j$, then we will prove that S contains some other colour $k \neq j$ and no point in a δ^{tiny}-neighbourhood of $\mathbf{x}$ activates the k-th blanket-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 δ^T of the twisted tunnel is smaller than any other inverse-polynomial quantities of interest, apart from δ^{tiny}, which itself is smaller than all other inverse-polynomials of interest, including δ^T. We provide the following definition of a *consistent colour*.

Definition 24 (Consistent Colour) *For $\mathbf{x} = (\tau; \alpha_2, \ldots, \alpha_n)$ in the Significant Region, colour $j \in \{\pm 2, \ldots, \pm n\}$, let $A_j \in \{A_+, A_-\}$ be the label that tends to increase in interval $[j-2, j]$ when the $|j|$-th coordinate α_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 colour *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.*

Condition 1 says that at $\mathbf{x} \in R(S)$, colour j is a member of S and the corresponding transformed coordinate is sufficiently far from the twisted tunnel. Condition 2 says that we are at least some (small but significant) distance from triggering the j-th blanket-sensor in a direction that corresponds to excessive colour $-j$. In other words, for the j-th blanket-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 colours exist.

Proposition 6.5 *Suppose* $\mathbf{x} = (\tau; \alpha_2, \ldots, \alpha_n)$ *belongs to outer region* $R(S)$ *and that* $\mathbf{x}$ *is within distance* δ^{tiny} *of the boundary of the Significant Region. Then* $\mathbf{x}$ *has a consistent colour in* $\{\pm 2, \ldots, \pm n\}$.

Proof. Let $\ell \in \arg\max_{i \in \{2,\ldots,n\}} |\alpha_i|$ be the index of a transformed coordinate with maximum absolute value. We may assume there is an inverse-polynomial quantity δ^+ such that for any point $\mathbf{x} = (\tau; \alpha_2, \ldots, \alpha_n)$ within distance δ^{tiny} of the boundary of the Significant Region, we have $|\alpha_\ell| \leq \delta^+$. Moreover, the width δ^T of the twisted tunnel is chosen to be much smaller that δ^+ (by an inverse-polynomial amount) but much larger than δ^{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^τ. Recall that $A_j \in \{A_+, A_-\}$ denotes the label that increases in the c-e region when we move in direction d_j^τ. Also, let $A_{-j} \in \{+, -\}$, $A_{-j} \neq A_j$, be the complementary label. For $j > 0$, this involves cuts j and $j-1$ moving away from each other; for $j < 0$, this involves them moving towards 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^τ causes cuts $j-1$ and j to move away from each other; this is illustrated in Figure 13.

We claim that j is a consistent colour for $\mathbf{x}$. Note first that $\alpha_j > 2\delta^T$ and therefore Condition 1 is satisfied, since $j > 0$ in this case. $\alpha_j > 2\delta^T$ follows from the fact that $j \in \arg\max_{i \in \{2,\ldots,n\}} |\alpha_i|$, 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 Condition 2, recall first that $\mathbf{x}$ has transformed coordinates $(\tau; \alpha_2, \ldots, \alpha_n)$, and that the origin of D_τ has transformed coordinates $\mathbf{0}_\tau = (\tau; 0, \ldots, 0)$. 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 13. Near the axis, where the cuts are evenly-spaced (see Proposition 4.4), movement in direction d_j^τ 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 α_j.

First, consider moving from $\mathbf{0}_\tau$ to point $\mathbf{x}_j = (\tau; 0, \ldots, 0, \alpha_j, 0, \ldots, 0)$ 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 13. Now consider moving from $\mathbf{x}_j$ to $\mathbf{x}$. In this process, the $(j-1)$-st cut 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 $j \in \arg\max_{i \in \{2,\ldots,n\}} |\alpha_i|$, it follows that $\alpha_{j-1} \leq \alpha_j$ and $\alpha_{j+1} \leq \alpha_j$. Then, there is a sub-interval 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 coloured entirely A_j, implying Condition 2. Overall, we obtain that j is a consistent colour.

Case 2: $j < 0$ (i.e., $\alpha_j < 0$). In this case, moving in direction d_j^τ causes cuts $|j|-1$ and $|j|$ to move towards each other; this is illustrated in Figure 14.

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 δ^w, which by Proposition 4.4 is smaller than $1/2n$ by an inverse-polynomial amount. This means that if we start at $\mathbf{0}_\tau$ and re-set 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 13, 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 colour 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 14; 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_j^τ.

Moving in direction d_j^τ causes an increase of the label A_j in the interval $[|j|-2, |j|]$. For j not to be a consistent colour, 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 14):

- 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 towards each other and therefore importantly for us here, cut $|j|-1$ to move to the left (towards integer point $|j|-2$).
- By increasing the transformed coordinate α_{j+1} in the negative direction, moving in direction $d^\tau_{-(|j|+1)}$. This causes cuts $|j|$ and $|j|+1$ to move towards 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 *right-hand-side* of $[|j|-2, |j|]$. We consider two more cases, depending on whether or not this is the case.

Case 2b(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_j^τ 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 $\arg\max_{i\in\{2,\dots,n\}} |\alpha_i|$, it is easy to verify that

- 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_j^τ (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_j^τ (from $\mathbf{0}_\tau$ to $\mathbf{x}_j$).

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

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

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 towards 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^τ_j.

First, we verify that $-(|j|+1)$ satisfies Condition 1 of Definition 24, i.e. that $\alpha_{|j|+1} < -2\delta^T$ (at this point we know that α_{j+1} is a negative quantity). We consider two cases, depending on whether τ 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 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^τ_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 τ 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 Condition 1 is satisfied for colour $-(|j|+1)$.

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 blanket-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 colour $-(|j|+1)$ is a consistent colour 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^{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 case that happens, we move on to consider interval $[j, j+2]$ and we apply the same argument. Again, α_{j+2} is negative, and since cut $|j|+2$ is to the left of location $j+1$ by a margin $\frac{p^{\text{large}}}{2p^{\text{huge}}} < 2\delta^T$, $-(|j|+2)$ satisfies Condition 1 to be a consistent colour. It will also satisfy Condition 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 colour 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 colour (as otherwise we would have identified a consistent colour in some already examined interval). But then $-n$ is a consistent colour, since we have moved an inverse-polynomial distance (at least $\frac{p^{\text{large}}}{2p^{\text{huge}}} < 2\delta^T$) in direction d^τ_{-n} (Condition 1), and at least $1/2$ of the interval $[n-2, n]$ is coloured in a way that agrees with this (Condition 2). □

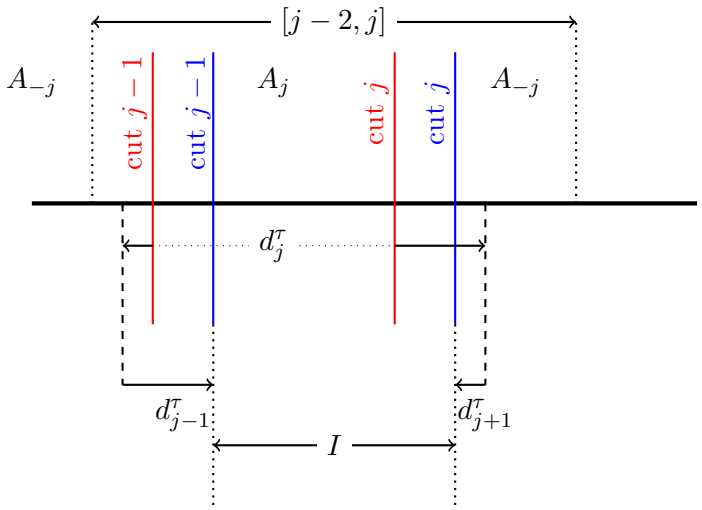


Figure 13: Illustration for Proposition 6.5, Case 1. For $i \in [n]$, cut i denotes the i-th (c-e) cut from the left. The cuts $j-1$ and j coloured red correspond to positions encoding part of $\mathbf{0}_\tau$. The dashed lines (to the left and to the right of the positions of the red cuts respectively) correspond to positions encoding part of point $\mathbf{x}_j$, after we have only moved α_j in direction d_j^τ. The cuts coloured blue correspond to positions resulting from subsequent movement in directions d_{j-1}^τ and d_{j+1}^τ, which encode part of $\mathbf{x}$. In the figure, the case where the average of the movements in d_{j-1}^τ, d_j^τ and d_{j+1}^τ forces both cuts to move to the right, compared to their original positions in the encoding of $\mathbf{0}_\tau$, is shown.

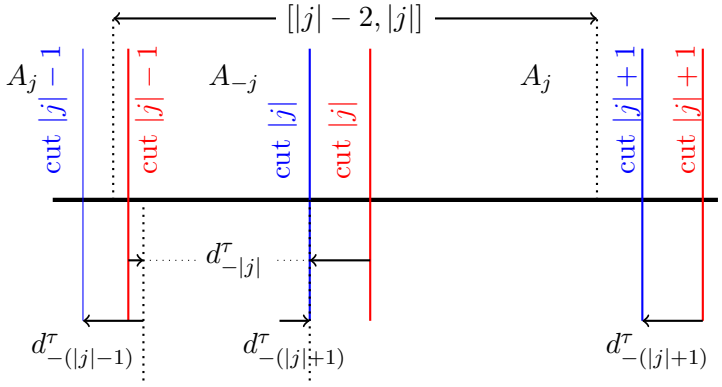


Figure 14: Illustration for Proposition 6.5, Case 2b(i). In this case, j is negative and therefore we use $|j|$ to represent the j-th cut from the left. Moving in direction d_j^τ causes cuts $|j|-1$ and $|j|$ to move towards each other. Again, the red cuts correspond to part of $\mathbf{0}_\tau$ and the blue cuts correspond to the encoding of part of $\mathbf{x}$, after averaging over the movement in directions $d_{-(|j|-1)}^\tau$, $d_j = d_{-|j|}$ and $d_{-(|j|+1)}^\tau$. 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 $|j|+1$ does not intersect the interval $[|j|-2, |j|]$ and therefore there is no additional amount of A_{-j} introduced to the right-hand side 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.

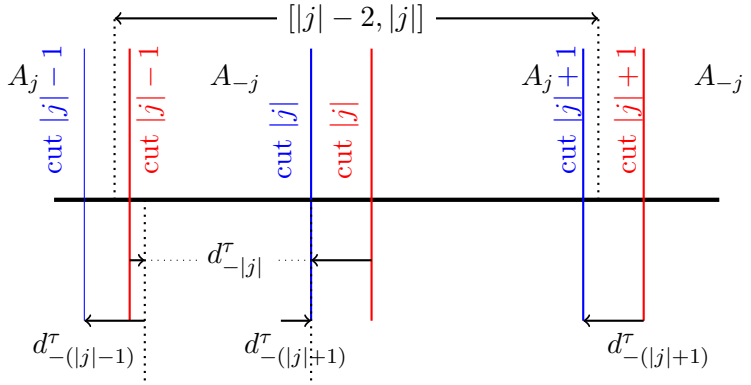


Figure 15: Illustration for Proposition 6.5, Case 2b(ii). In this case, j is negative and therefore we use $|j|$ to represent the j-th cut from the left. Moving in direction d^τ_j causes cuts $|j|-1$ and $|j|$ to move towards each other. Again, the red cuts correspond to part of $\mathbf{0}_\tau$ and 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}. 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 right-hand side of $[|j|-2,|j|]$. By the argument of Case 2b(ii), either $-(|j|+1)$ will be a consistent colour, 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 colour.

Corollary 6.6 *A solution S_{CH} to I_{CH} cannot encode a point $\mathbf{x}$ in the Significant Region, within distance δ^{tiny} of the boundary (where blanket-sensor agent(s) become active).*

Proof. Observation 6.1 tells us that the k-th component of f' is the difference between A_+ and A_- observed by c-e agent a_k, and Proposition 6.2 tells us that all these components, averaged over a set of points within δ^{tiny} of $\mathbf{x}$ need to be close to zero, at a solution.

Proposition 6.5 tells us that $\mathbf{x}$ has some consistent colour k. All points within δ^{tiny} of $\mathbf{x}$ cause two outputs (gates g'_k, g'_{-k} as defined in Section 5.1.2) of the circuit-encoders to represent colour k. This includes points where blanket-sensor agents are active, since by the properties of consistent colours, we are at least an inverse-polynomial distance from any point where any $b_{i,k}$ can be active in the wrong direction. In Section 5.1.3 the blanket-sensor agents are designed to agree with the definition of consistent colour, Definition 24. So c-e 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 6.7 *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 5.3. The points $\mathbf{x}$ and $\mathbf{x}'$ have been coloured according to Item 2 of Section 5.3, and they belong to two long thin colour-regions that extend the cubelets that lie on the panchromatic facets of the cube embedded at the centre 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 colours are equal and opposite. □

Remark: The $\mathbf{x}$, $\mathbf{x}'$ in Proposition 6.7 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 right-hand side of the c-e region. Proposition 6.7 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 6.7 tells us that colour-regions are adjacent to colour-regions having the opposite colour. We need to verify that for a pair $\mathbf{x}, \mathbf{x}'$ of points that are close together but have opposite colours (due to lying in such a pair of colour-regions) the same (and not opposite) feedback is provided to the c-e agents. (So, in contrast with a pair of opposite-colour points that represent a solution, whose feedback to the c-e 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 blanket-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 over-represented label is fed back to c-e agent a_j).

In the case that no blanket-sensor agents are active, if $\mathbf{x}, \mathbf{x}'$ receive opposite colours 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 the right-hand side of the c-e region, and in moving to $\mathbf{x}'$ we move that cut to the left-hand side. 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 re-invert this feedback by reversing the colour, and hence the output of f'. This is very similar (in fact, an n-dimensional analogue) to the handling of the "wrap-around points" in the sequence via the interpretation in terms of the virtual cuts in [29].

7 Further work

What is the computational complexity of k-thief Necklace-splitting, for k not a power of 2? As discussed in [56, 49], 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 [33].

Interestingly, Papadimitriou in [60] (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 [60]. Given the discussion above, it could possibly be the case that the the principle associated with Necklace-splitting for k-thieves is the PPA-k principle instead.

What about the computational hardness of the problem? Is 3-thief Necklace-splitting hard for PPA? 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 is unlikely. However, it is worth mentioning here that the computational

equivalence between ε-Consensus-halving and Necklace-splitting that was proven in [29] is actually established between the Necklace Splitting problem for any k and the corresponding approximate $1/k$-Division problem, a generalization of ε-Consensus-halving (see [66]); a PPA-hardness or PPA-membership result for $k > 2$ for the latter problem would imply a corresponding result for Necklace-splitting with $k > 2$. En route to either of these results, a possible approach that was suggested in [66], would be to define an appropriate generalisation of Tucker's Lemma and prove it *constructively* (see Section 8 in [66]).

We have left open the questions of whether ε-Consensus-halving remains PPA-complete for constant ε, 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 [28]; 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 [6, 33]; note also the related work on Bounded Arithmetic discussed in [33]. 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, [54] initiates an interesting experimental study of path-following algorithms for 2-thief Necklace-splitting, obtaining positive results when the number of bead colours is not too large. However, path-following seems to be inapplicable for, say, 3 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.

Acknowledgements We thank Alex Hollender for detailed and insightful proof-reading of earlier versions of this paper.

References

[1] J. Aisenberg, M.L. Bonet, and S. Buss. 2-D Tucker is PPA complete. *Electronic Colloquium on Computational Complexity*, TR15-163 (2015).

[2] N. Alon. Splitting Necklaces. *Advances in Mathematics*, **63**(3), pp. 247–253 (1987).

[3] N. Alon. Some recent Combinatorial Applications of Borsuk-type Theorems. *London Mathematical Society Lecture Notes* Series 131, Algebraic, Extremal and Metric Combinatorics 1986, pp. 1–12. CUP (1988).

[4] N. Alon. Non-Constructive Proofs in Combinatorics. *Procs. of International Congress of Mathematicians 1990 (Kyoto, Japan)*, pp. 1421–1429. Tokyo, Springer-Verlag (1991).

[5] N. Alon and D.B. West. The Borsuk-Ulam theorem and bisection of necklaces. *Proceedings of the American Mathematical Society*, **98**(4), pp. 623–628 (1986).

[6] P. Beame, S. Cook, J. Edmonds, R. Impagliazzo and T. Pitassi. The relative complexity of NP search problems. *Journal of Computer and System Sciences* **57**, pp. 3–19 (1998).

[7] A. Belovs, G. Ivanyos, Y. Qiao, M. Santha, and S. Yang. On the Polynomial Parity Argument complexity of the Combinatorial Nullstellensatz. *Procs. of the 32nd Computational Complexity Conference*, Article No. 30 (2017).

[8] S.N. Bhatt and C.E. Leiserson. How to Assemble Tree Machines (Extended Abstract). *Procs. of the 14th annual ACM Symposium on Theory of Computing (STOC)*, pp. 77–84 (1982).

[9] P.V.M. Blagojević and P. Soberón. Thieves can make sandwiches. *Bull. of the London Mathematical Society*, 50(1) pp. 108–123 (2018).

[10] P. Bonsma, Th. Epping, and W. Hochstättler. Complexity results on restricted instances of a paint shop problem for words. *Discrete Applied Mathematics*, 154, pp. 1335–1343 (2006).

[11] S.R. Buss and A.S. Johnson. Propositional proofs and reductions between NP search problems. *Annals of Pure and Applied Logic*, 163, pp. 1163–1182 (2012).

[12] X. Chen, D. Dai, Y. Du, and S.-H. Teng. Settling the Complexity of Arrow-Debreu Equilibria in Markets with Additively Separable Utilities. *Procs. of FOCS*, Atlanta, USA, pp. 273–282 (2009).

[13] X. Chen and X. Deng. On the complexity of 2D discrete fixed point problem. *Theoretical Computer Science*, **410** (44) pp. 4448–4456 (2009).

[14] X. Chen, X. Deng, and S.-H. Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM* **56**(3) pp. 1–57 (2009).

[15] X. Chen, D. Durfee, and A. Orfanou. On the Complexity of Nash Equilibria in Anonymous Games. *Procs. of 47th STOC*, Portland, USA, pp. 381–390 (2015).

[16] X. Chen, D. Paparas, and M. Yannakakis. The complexity of non-monotone markets. *Journal of the ACM*, **64**(3), article 20 (2017).

[17] Y. Chen, J.K. Lai, D.C. Parkes, and A.D. Procaccia. Truth, justice, and cake cutting. *Games and Economic Behavior*, **77**(1) pp. 284–297 (2013).

[18] B. Codenotti, A. Saberi, K. Varadarajan, and Y. Ye. The complexity of equilibria: Hardness results for economies via a correspondence with games. *Theoretical Computer Science*, **408** pp. 188–198 (2008).

[19] C. Daskalakis, P.W. Goldberg, and C.H. Papadimitriou. The Complexity of Computing a Nash Equilibrium. *SIAM Journal on Computing* **39**(1) pp. 195–259 (2009).

[20] C. Daskalakis, and C.H. Papadimitriou. Continuous Local Search. *2nd ACM Symposium on Discrete Algorithms* (2011).

[21] C. Daskalakis, C. Tzamos, and M. Zampetakis. A converse to Banach's fixed point theorem and its CLS-completeness. *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, 2018.

[22] X. Deng, J.R. Edmonds, Z. Feng, Z. Liu, Q. Qi, and Z. Xu. Understanding PPA-completeness. *31st Conference on Computational Complexity*, article no. 23, pp. 23:1–23:25 (2016).

[23] X. Deng, Z. Feng, and R. Kulkarni. Octahedral Tucker is PPA-complete. *Electronic Colloquium on Computational Complexity* Report TR17-118 (2017).

[24] X. Deng, Q. Qi, and A. Saberi. On the Complexity of Envy-Free Cake Cutting. *CoRR*, arXiv:0907.1334 (2009).

[25] H. Edelsbrunner and R. Waupotitsch. Computing a Ham-Sandwich Cut in Two Dimensions. *J. Symbolic Computation*, **2**(2) pp. 171–178 (1986).

[26] E. Elkind, L.A. Goldberg and P.W. Goldberg. Nash Equilibria in Graphical Games on Trees Revisited. *Procs. of the 7th ACM Conference on Electronic Commerce (ACM EC)*, Ann Arbor, Michigan, USA, pp. 100–109 (2006).

[27] J. Fearnley, S. Gordon, R. Mehta, and R. Savani. End of Potential Line. *arXiv preprint arXiv:1804.03450,* 2018.

[28] A. Filos-Ratsikas, S. K. S. Frederiksen, P.W. Goldberg, and J. Zhang. Hardness Results for Consensus-Halving. *Procs. of the 43rd MFCS Symposium (MFCS 2018)*, pp. 24:1–24:16 (2018).

[29] A. Filos-Ratsikas and P.W. Goldberg. Consensus-Halving is PPA-Complete. *Procs. of the 50th STOC*, pp. 51–64 (2018).

[30] R.M. Freund and M.J. Todd. A Constructive Proof of Tucker's Combinatorial Lemma. *Journal of Combinatorial Theory* Series A **30**, pp. 321–325 (1981).

[31] S. Garg, O. Pandey, and A. Srinivasan. On the exact cryptographic hardness of finding a Nash equilibrium. *Cryptology ePrint Archive*, Report 2015/1078 (2015).

[32] S. Garg, O. Pandey, and A. Srinivasan. Revisiting the Cryptographic Hardness of Finding a Nash Equilibrium. *Procs. of CRYPTO*, Part II, LNCS 9815, pp. 579–604 (2016).

[33] P.W. Goldberg and C.H. Papadimitriou. Towards a Unified Complexity Theory of Total Functions. *Journal of Computer and System Sciences*, **94**, pp. 167–192 (2018).

[34] C.H. Goldberg and D.B. West. Bisection of Circle Colorings. *SIAM Journal on Algebraic Discrete Methods* **6** (1) pp. 93–106 (1985).

[35] F. Grandoni, R. Ravi, M. Singh, and R. Zenklusen. New approaches to multi-objective optimization. *Math. Program. Ser. A* **146** pp. 525–554 (2014).

[36] M. Grigni. A Sperner Lemma Complete for PPA. *Information Processing Letters* **77** (5–6) pp. 255–259 (2001).

[37] T.P. Hill. Determining a Fair Border. *The American Mathematical Monthly*, **90**(7) pp. 438–442 (1983).

[38] C.R. Hobby and J.R. Rice, A Moment Problem in L_1 Approximation. *Proc. Amer. Math. Soc.* 16(4) pp.665–670 (1965).

[39] P. Hubáček, and E. Yogev. Hardness of continuous local search: Query complexity and cryptographic lower bounds. *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms*, 2018.

[40] E. Jeřábek. Integer factoring and modular square roots. *Journal of Computer and System Sciences* **82**(2) pp. 380–394 (2016).

[41] D.S. Johnson, C.H. Papadimitriou, and M. Yannakakis. How easy is local search? *J. Comput. System Sci.* **37** pp. 79–100 (1988).

[42] C.S. Karthik and A. Saha. Ham Sandwich is Equivalent to Borsuk-Ulam. *Symposium on Computational Geometry* pp. 24:1–24:15 (2017).

[43] S. Kintali, L.J. Poplawski, R. Rajaraman, R. Sundaram, and S.-H. Teng. Reducibility among Fractional Stability Problems. *SIAM Journal on Computing*, **42**(6) pp. 2063–2113 (2013).

[44] C. Knauer, H.R. Tiwary, D. Werner. On the computational complexity of Ham-Sandwich cuts, Helly sets, and related problems. *Procs of STACS* pp. 649–660 (2011).

[45] I. Komargodski, M. Naor, and E. Yogev. White-Box vs. Black-Box Complexity of Search Problems: Ramsey and Graph Property Testing. *Electronic Colloquium on Computational Complexity* Report TR17-015 (2017).

[46] C-Y Lo, J. Matoušek, and W. Steiger. Ham-sandwich cuts in R^d. *Procs of the 24th STOC*, pp. 539–545 (1992).

[47] Chi-Yuan Lo, Jiri Matoušek, and William Steiger. Algorithms for Ham-Sandwich Cuts. *Discrete and Computational Geometry* **11**(4) pp. 433–452 (1994).

[48] J.A. De Loera, X. Goaoc, F. Meunier, and N. Mustafa. The discrete yet ubiquitous theorems of Carathéodory, Helly, Sperner, Tucker, and Tverberg. *CoRR*, arXiv:1706.05975 (2017).

[49] M. de Longueville and R.T. Živaljević. The Borsuk-Ulam-property, Tucker-property and constructive proofs in combinatorics. *Journal of Combinatorial Theory, Series A* 113 pp. 839–850 (2006).

[50] J. Matoušek. Geometric range Searching. *ACM Computing Surveys.* **26**(4) pp. 421–461 (1994).

[51] Matoušek, Jiří. Lectures on discrete geometry. Vol. 212. New York: Springer, 2002.

[52] J. Matoušek. *Using the Borsuk-Ulam Theorem.* Lectures on Topological Methods in Combinatorics and Geometry, Springer (2008).

[53] F. Meunier. Discrete Splittings of the Necklace. *Mathematics of Operations Research*, 33, No. 3, pp. 678–688 (2008).

[54] F. Meunier and B. Neveu. Computing solutions of the paintshop-necklace problem. *Computers and Operations Research*, 39(11), pp. 2666–2678 (2012).

[55] F. Meunier and A. Sebő. Paintshop, odd cycles and necklace splitting. *Discrete Applied Mathematics*, 157, pp. 780–793 (2009).

[56] F. Meunier. Simplotopal maps and necklace splitting. *Discrete Mathematics*, 323, pp. 14–26 (2014).

[57] N. Megiddo. A note on the complexity of P-matrix LCP and computing an equilibrium. *Res. Rep. RJ6439, IBM Almaden Research Center, San Jose.* pp. 1–5 (1988).

[58] N. Megiddo and C.H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, **81**(2) pp. 317–324 (1991).

[59] R. Mehta. Constant rank bimatrix games are PPAD-hard. *Procs. of 46th STOC*, New York, USA, pp. 545–554 (2014).

[60] C.H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences* **48** pp. 498–532 (1994).

[61] P. Pudlák. On the complexity of finding falsifying assignments for Herbrand disjunctions. *Arch. Math. Logic*, 54, pp. 769–783 (2015).

[62] S. Roy and W. Steiger. Some Combinatorial and Algorithmic Applications of the Borsuk-Ulam Theorem. *Graphs and Combinatorics*, 23, Supplement 1, pp. 331–341 (2007).

[63] A. Rubinstein. Inapproximability of Nash equilibrium. *Procs. of the 47th STOC*, Portland, USA, pp. 409–418 (2015).

[64] A. Rubinstein. Settling the complexity of computing approximate two-player Nash equilibria. *Procs. of the 57th FOCS*, New Brunswick, USA, pp. 258–265 (2016).

[65] S. Schuldenzucker, S. Seuken, and S. Battiston. Finding clearing payments in financial networks with credit default swaps is PPAD-complete. *Proceedings of the 8th Innovations in Theoretical Computer Science (ITCS) Conference*, Berkeley, USA, (2017).

[66] F.W. Simmons and F.E. Su. Consensus-halving via theorems of Borsuk-Ulam and Tucker. *Mathematical Social Sciences* 45(1) pp. 15–25, (2003).

[67] . K. Sotiraki, M. Zampetakis, and G.Zirdelis. PPP-Completeness with Connections to Cryptography. *Proceedings of the 59th IEEE Symposium on Foundations of Computer Science*, 2018.

[68] A. H. Stone and J.W. Tukey. Generalized "sandwich" theorems. *Duke Mathematical Journal*, **9**: pp. 356–359, (1942).

[69] A. Thomason. Hamilton cycles and uniquely edge colourable graphs. *Ann. Discrete Math.* **3**, pp. 259–268 (1978).

[70] A.W. Tucker. Some topological properties of disk and sphere. *Proc. First Canadian Math. Congress*, Toronto: University of Toronto Press, pp. 285–309 (1945).

[71] V.V. Vazirani and M. Yannakakis. Market equilibrium under separable, piecewise-linear, concave utilities. *Journal of the ACM*, **58** (3), Article 10 (2011).

APPENDIX

A Details from [29].

In this section, we include several details from [29] that are being used (or extended) in the present paper as well.

Bit detection gadgets

First, the ability to detect the position of the cuts in the c-e region and feed this information to the circuit lies in the presence of gadgets developed in [28] and used in [29], referred to as "bit detection gadgets" in [29]. 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 16 or Figure 20, top). 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 the favour of one against the other, there will be a cut 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, e.g.,

- at the value-blocks of the sensor agents (Definition 13) in the region R_i (see Figure 16) that assumes the right or left position depending on whether their small value-blocks in the c-e region are labelled A_+or A_-,
- in the encoding of the circuit C_{VT} of New variant high-D Tucker using the circuit-encoding agents C_i (also see Section 5.1),
- at the value-blocks of the blanket-sensor agents (Definition 14) in the region R_i, with the difference that in that case, a small value-block (of value $9\kappa/10$) lies between the two thin and dense valuation blocks of the bit detection gadget (see Figure 17).

Sensor and blanket sensor agents

The formal definitions of the sensor agents and the blanket sensor agents were given in the main text, see Definition 13 and Definition 14 respectively. Here, we explain in more detail how these agents make use of the bit-detection gadgets (which lie in the region R_i) to detect the positions of the cuts (for the sensor agents) or to detect large discrepancies on the volumes of the two labels in the c-e region (for the blanket sensor agents).

Starting from the sensor agents, recall that each such agent of $S_i \subset C_i$ 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 sub-region R_i (recall that the circuit-encoding region R is partitioned into sub-regions 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, as described in the previous subsection - see Figure 20, top, for an illustration. If the value-block on the left-hand side (in the c-e region) is labelled A_-, then the cut on the right-hand side 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 pre-processing unit of the circuit (see Section 5.1.1) and

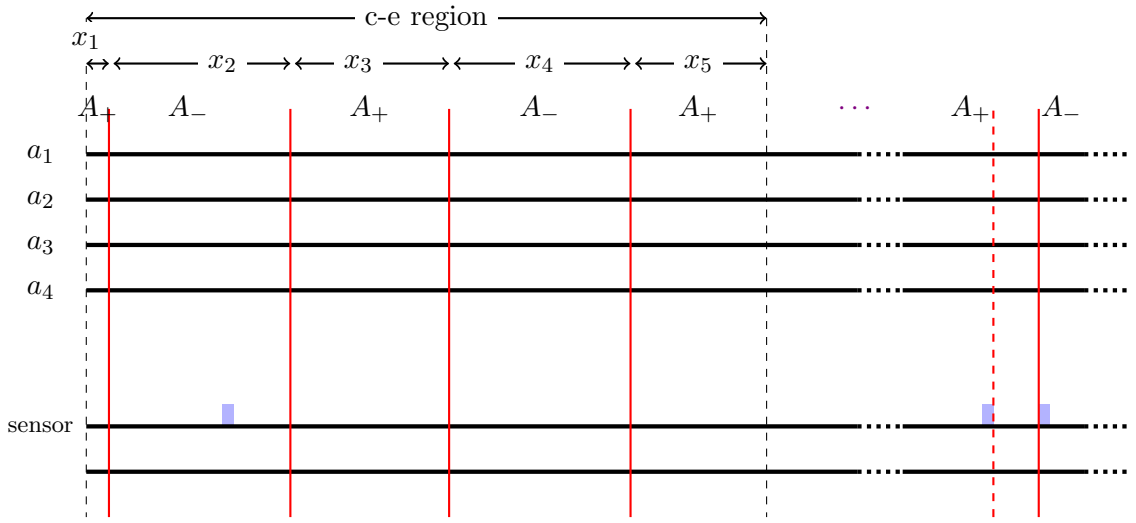


Figure 16: An example of how the input of a sensor agent is processed into a boolean value that will be used by the encoding of the circuit. On of the two value-blocks on the right-hand side of the picture (the bit-detection gadget) is interesected by the cut corresponding to the sensor, depending on whether the value-block on the left-hand side is labelled A_+or A_-. In the figure, the block is labelled $A_+$and therefore the cut intersects the rightmost value-block on the right-hand side. The other (unused) option is depicted by a red dashed ine.

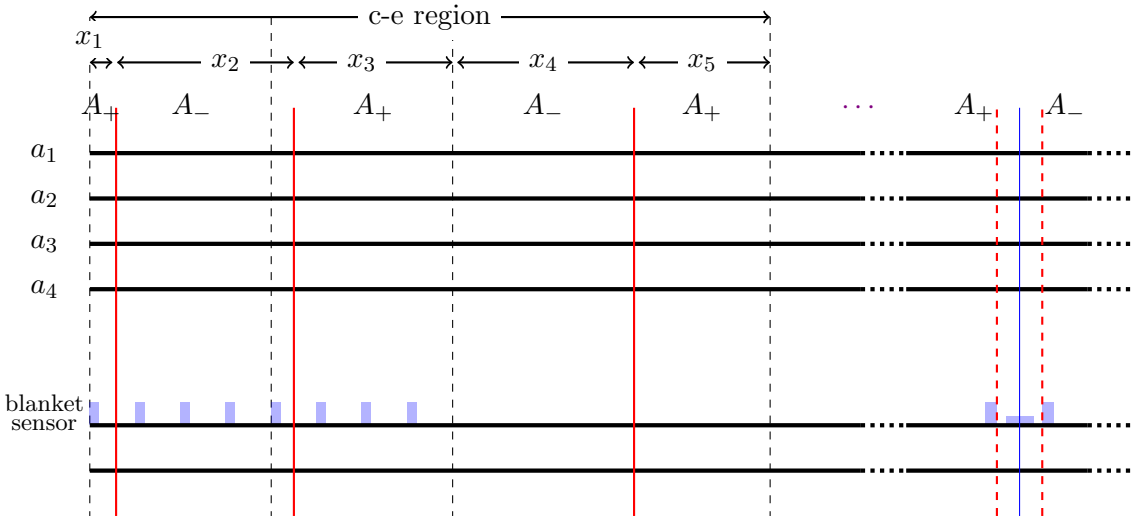


Figure 17: An example on how the blanket-sensor agents provide input to the circuit for their monitored intervals. Depending on the balance in labels for the value-blocks on the left-hand side, the bit-detection gadget of the blanket-sensor agent on the right-hand side assumes the leftmost, middle, or rightmost positions. In the particular example, the number of value-blocks on the left-hand side for each label is balanced, and therefore the cut on the right-hand side (shown in blue) interesects the middle value-block of the bit-detection gadget. In this case, the blanket-sensor is not active. The other two possible positions for the cut on the right-hand side, when the blanket sensor is active, are depicted by dashed red lines.

the subsection following this one. In [29], we referred to this information as "raw data" (although the cut extraction mechanism there had to be more elaborate, due to the inversely-exponential precision). The pre-processing unit is responsible for converting the raw data into appropriate inputs for the circuit C_{VT}, which encode coordinate of points on the Möbius-simplex. 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 th following subsection.

The blanket sensor agents use very similar bit detection gadgets 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-blocks (the block of value $9\kappa/10$ in Definition 14). This is because the blanket sensor agents needs 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 blanket 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 over-represented label and the parity of the index of the blanket sensor agent. It is straightforward (as before) to interpret these positions as boolean values. The main idea in [29] is that if the blanket sensor agents are active, then this information will "over-ride" the circuit C_{VT} and generate an imbalance of labels in the feedback provided to the c-e agents directly, essentially bypassing the output gates of C_{VT}. We use the same principle here, but since we now have many blanket sensor agents, extra care must be taken to make sure that no artificial solutions are introduced when colouring the domain. The details on how the input from the blanket-sensors affects the feedback to the c-e agents are presented in Section 5.1.3.

Encoding of the circuit C_{VT} of New variant high-D Tucker.

Before we explain how the circuit of New variant high-D Tucker is encoded using the circuit encoders C_i, we present the main building blocks for simulating boolean circuits based on bit-detection gadget, first presented in [28] and later adapted and used in [29].

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 gate gadgets, shown in Figure 18 (from [29]).

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. Also note that for the inputs, as well as the output of the NOT gate, the label on the left-hand side of the cut is A_+ and the label on the right-hand side will be A_-, whereas for the outputs to the OR and AND gate, the label on the left-hand side of the cut is A_- and the label on the right hand side is A_+ . This can be achieved with the appropriate use of parity gadgets, i.e., simple valuation blocks that force cuts to lie in specific positions, only to change the parity of the cut sequence (see [29] for more details).

As we mentioned earlier, the bit-detection gagdets allow us to extract boolean values corresponding to the positions of the cuts in the c-e region - this is achieved via the use of the sensor agents. In the next step, these values (referred earlier as the "raw data") are supplied into a gadget that is referred to as the pre-processing circuit in [29]. The role of this circuit is to convert this information to coordinates of points on the Möbius-simplex, which then will go through a coordinate transformation (see Section 5.2) and will be used as inputs to the encoding of C_{VT}. The pre-processing circuit can be implemented using the boolean gate gadgets described above. The same is true for the actual circuit C_{VT} as well, which can be simulated using the same set of gadgets, using the principle described in Figure 19 (from [29]). The outputs of the pre-preprocessing

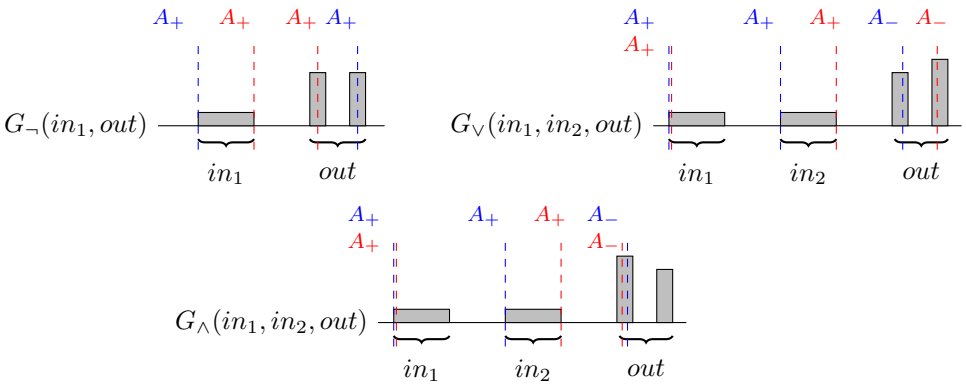


Figure 18: The Boolean Gate Gadgets encoding the NOT, OR and AND gates. For visibility, the valuation blocks are not according to scale. For the NOT gate, the input has value 0.25 and the output blocks have volume 0.375 each. For the OR (respectively AND) gate, the input blocks have value 0.125 each 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 colour, and the labels on the left-side of these cuts are shown and colour-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 colour. For the OR gate, when both inputs cut 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 left-hand side 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).

circuit are inputed to the input gates of C_{VT} and their outputs, in turn, are inputted to the gates on the next level and this process carries on until the output gates of C_{VT}.

Unreliable circuits and the stray cut

In the main text, we mentioned that we are guaranteed that at most n cuts will lie in the c-e region (Observation 4.1) and at least $n-1$ cuts will lie in the c-e region, as otherwise some blanket sensor agent would be active (Proposition 4.4). If $n-1$ cuts lie in the c-e region, this means that one of the n c-e cuts has moved into the circuit encoding region; following [29], we will refer to that cut as a *stray cut*. As we highlighted in [29], the presence of a stray cut may have the following two consequences.

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 left-hand side, it now sees A_- and vice-versa.

The first effect is not much of a problem, we will simply assume that the corresponding circuit is unreliable. As explained in the main text, an unreliable circuit can have an arbitrary (or even adversarial) effect on the volume of $A_+$and A_-supplied as feedback to the c-e agents, but since there will be at most 1 unreliable circuit, its effect will be "outvoted" by the remaining reliable

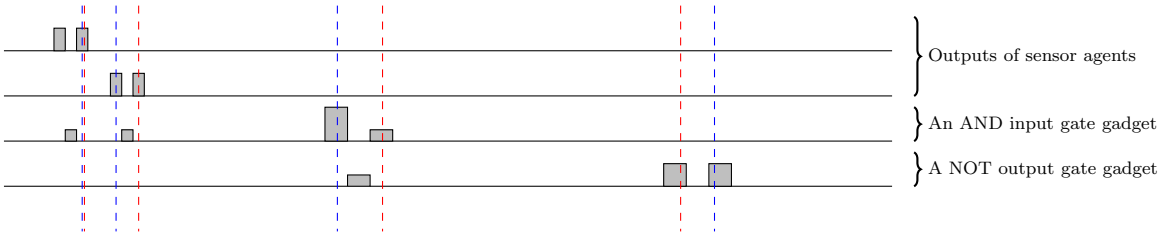


Figure 19: The basic idea behind the gate-agents encoding the gates of C_1. The picture denotes a simplified case where two input bits from the bit-encoders are supplied to an enconding of an input AND gate of C_i and the output bit of this gate is in turn supplied to the encoding of a NOT output gate 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 sit 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.

circuits (circuits that do not receive reliable inputs will also be outvoted by those that do, since there are at most n of them, see Observation 4.2).

The parity flip that the stray cut induces on other circuit-encoders however seems potentially more worrisome., since it could flip the output of the sensor agents that detect the positions of the cuts. This is taken care of by the presence of the reference sensor agent (Definition 21) which achieves the disorientation of the domain. As explained in Section 5.1.1, the outputs of circuit C_i are taken with reference to the value $s_{1,1}$ of the reference senso agent, in the sense that after simulating C_{VT} we take the exclusive-or with $s_{1,1}$; this can be implemented in the circuit using the boolean gate gadgets explained above, in a manner similar to [29] (via the use of the XOR sub-circuit, see Section 4.4.2 in [29]). This allows us to use a similar "double-negative lemma" as the one we used in [29] (see Lemma 5.4).

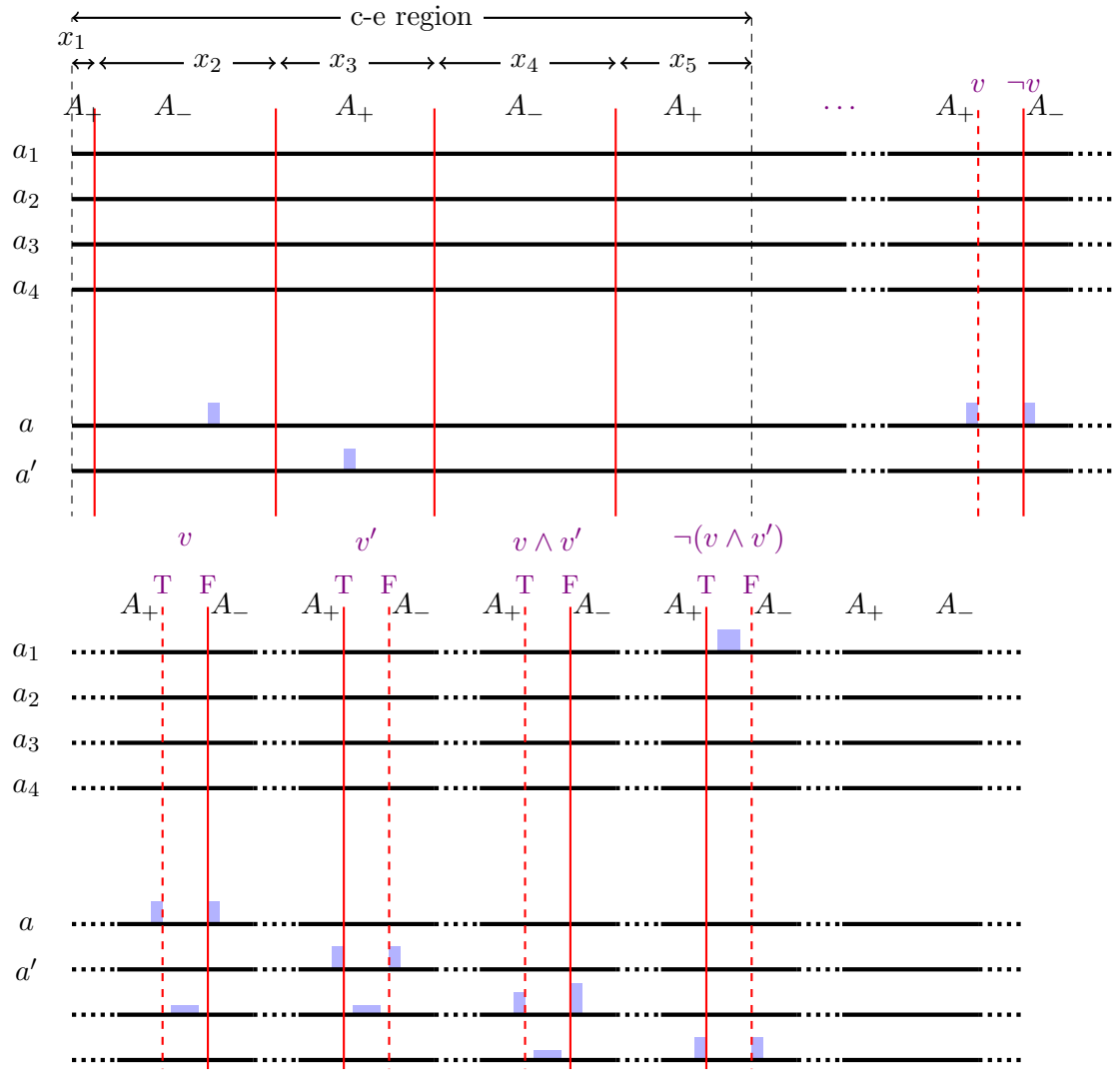


Figure 20: Example showing gate simulation: $n = 4$; agents a and a' have corresponding propositional variables v and v' that are two inputs to a circuit. $v =$ FALSE since a's sensor-value lies in a region labelled A_-, similarly $v' =$ TRUE since a''s sensor-value lies in a region labelled A_+. Gate-encoding 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-e agent a_1 receives feedback based on the conjunction of 2 input bits.

B Membership in PPA

We show that Discrete Ham Sandwich belongs to PPA, which appears to be folklore that has not, to out knowledge, been written down. (In fact, in [60] the problem was claimed to belong to PPAD, which is now seen to be incorrect subject to PPAD $\neq$ PPA, by the result in [1]). Since Theorem 2.3 reduces from 2-thief Necklace-splitting to Discrete Ham Sandwich, it follows immediately that 2-thief Necklace-splitting belongs to PPA. In Section B.2 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.

B.1 Discrete Ham Sandwich is in PPA

We use Freund and Todd's [30] construction of an undirected graph with known degree-1 vertex, based on a "special triangulation" of a high-dimensional L_1 ball (i.e. a high-dimensional octahedron, also known as the crosspolytope [52]). 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 give 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.)

Based on I we identify an exponentially-large collection of "candidate hyperplanes" as follows, which contains a solution to I. A candidate hyperplane H is represented by a gradient vector g_H whose coordinates are assumed to be normalised to 1 (L_1 norm; their absolute values sum to 1). Given g_H, H is obtained by using binary search to efficiently find a hyperplane with gradient g_H that bisects the union of the sets S_i; it is then easy to check whether H is a solution. Note that there exists an integer N whose binary expansion has length polynomial in $|I|$ such that entries of some solution g_H can be assumed to be multiples of $1/N$.

Fix a point $p \in \mathbb{R}^n$ such that p is not contained in any candidate solution to I, e.g. $p = (1/2N, 1, \ldots, 1)$. Given any H, the "positive" side of H is the side that contains p. Assume we chose N large enough so that if two hyperplanes H and H' have gradients g_H and $g_{H'}$ whose coordinates differ by at most $1/N$, then for any point x in I where they disagree, x lies in H or H' (and does not lie strictly on the positive side of one and the negative side of the other).

For any H, label it as follows. Find the set S_i that is most unevenly split by H breaking ties lexicographically. Label H with i if most of S_i lies on the positive side, otherwise label H with $-i$. Each hyperplane H has a g_H that is a point on the L_1-norm sphere S^{n-1}. By construction, antipodal points receive opposite labels. We can use these as the vertices of a special triangulation of the octahedral ball B^n, in which the origin is used as an additional point and is connected to all the points on S^{n-1} that correspond to candidate solutions. The graph defined in [30] is a degree 2 undirected graph with a single known degree-1 vertex (the origin), and for which any other degree-1 vertex represents a pair of hyperplanes that bisect all the S_i.

B.2 Necklace-splitting is in PPA, if the number of thieves is a power of 2

The version of the problem with two thieves, 2-thief Necklace-splitting, belongs to PPA since we reduced it to Discrete Ham Sandwich which is shown in Section B.1 to belong to PPA. We can extend the membership to PPA for k-Necklace-splitting, when k is a power of 2, by using the argument of Proposition 3.2 of Alon [2] (here, we take both k and l to be 2). In particular, we will reduce Necklace-splitting to 4-Necklace-splitting.

We start from an instance of 4-Necklace-splitting (with 4 thieves) and we regard it as an instance of Necklace-splitting (with 2 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 colour. 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 sub-intervals of the same collection together); note that each new instance will have an even number of beads of each colour, since we initial number of beads from each colour was a multiple of 4. Then we run the algorithm again on the resulting instances of Necklace-splitting to obtain a partition into 4 collections (2 for each individual instance), which consitutes a partition of the 4-Necklace-splitting into 4 collections according to the definition of the problem. If n is the number of colours, 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 Buss and Johnson [11], which shows that PPA and some related complexity classes are closed under Turing reductions.