ABSTRACT
We show that the computational problem CONSENSUS-HALVING is PPA-complete, the first PPA-completeness result for a problem whose definition does not involve an explicit circuit. We also show that an approximate version of this problem is polynomial-time equivalent to NECKLACE SPLITTING, which establishes PPAD-hardness for NECKLACE SPLITTING, and suggests that it is also PPA-complete.

CCS CONCEPTS
• Theory of computation → Complexity classes; Problems, reductions and completeness; Algorithmic game theory;

KEYWORDS
PPA-Completeness, Consensus-Halving, Necklace Splitting

1 INTRODUCTION
The class TFNP [32] of total search problems in NP (where every instance has an easily-checkable solution) does not seem to have complete problems. Moreover, no problem in TFNP can be NP-complete unless NP=co-NP. Consequently, alternative notions of computational hardness need to be developed and applied in our effort to understand the many and varied problems in TFNP that seem to be intractable.

The complexity class PLS (Johnson et al. [27]), and the classes PPAD, PPA, and PPP (Papadimitriou [34]) are subclasses of TFNP associated with various combinatorial principles that guarantee totality. Each principle has a corresponding definition of a computational problem whose totality applies that principle in the most general way possible, and a complexity class of problems reducible to it. In more detail:

• PLS consists of problems whose totality invokes the principle that every directed acyclic graph has a sink vertex;

• PPAD consists of problems whose totality is based on the principle that given a source in a directed graph whose vertices have in-degree and out-degree at most 1, there exists another degree-1 vertex;

• PPA differs from PPAD in that the graph need not be directed; being a more general principle, PPA is thus a superset of PPAD;

• PPP, based on the pigeonhole principle, consists of problems reducible to PIGEONHOLE CIRCUIT.

Of these complexity classes, so far only PLS and PPAD has succeeded in capturing the complexity of “natural” computational problems, and the main point of the present paper is to show for the first time that this is also true for PPA.

The Consensus-Halving problem involves a set of n agents each of whom has a valuation function on a 1-dimensional line segment A (the “cake”, in cake-cutting parlance). Consider the problem of selecting k “cut points” in A that partition A into k + 1 pieces, then labelling each piece either “positive” or “negative” in such a way that each agent values the positive pieces equally to the negative ones. In 2003, Simmons and Su [40] showed that this can always be done for k = n; their proof applies the Borsuk-Ulam theorem and is a proof of existence analogous to Nash’s famous existence proof of equilibrium points of games, proved using Brouwer’s or Kakutani’s fixed point theorem. Significantly, Borsuk-Ulam is the undirected version of Brouwer, and already from [34] we know that it relates to PPA, making Consensus-Halving a candidate for PPA-completeness. As detailed in Definition 2.1, we assume that valuations are presented as step functions using the logarithmic cost model of numbers.

1.1 Related Work
The complexity class PPAD has been successful in capturing the complexity of many versions of Nash equilibrium [9, 10, 14, 18, 33, 37] and market equilibrium computation [7, 11, 13, 39, 43], also cake-cutting [17]. Kintali et al. [28] extend PPAD-completeness to further domains including network routing, coalitional games, combinatorics, and social networks. Rubinstein [38] introduced an exponential-time hypothesis for PPAD to rule out a PTAS for approximate Nash equilibrium computation on bimatrix games. The class PLS represents the complexity of an even larger number of local optimisation problems. These results speak to the importance of PPAD and PLS as complexity classes. By contrast, hitherto the only problems known to be PPA-complete are ones that involve circuits (or equivalently, polynomial-time Turing machines) in their definition, which represented a critique of PPA. Noting that consensus-halving is a kind of social-choice problem, our result can be seen as an example of computational social-choice helping to populate “lonely” complexity classes, a phenomenon recently
reviewed by Hemaspaandra [24]. The complexity class PPP still suffers from that problem, although the present paper should raise our hope that problems such as \textsc{Equal Subsets} will turn out to be complete for PPP. Oracle separations of all these classes are known from [4].

The distinction between PPAD and PPA revolves around whether we are searching for a fixpoint in an oriented topological space, or an unoriented one. For example, while Papadimitriou [34] showed that it’s PPAD-complete to find a Sperner solution in a 3D cube, Grigni [23] showed that it’s PPA-complete to find a solution to Sperner’s lemma in a 3-manifold consisting of the product of a Möbius strip and a line segment. The 2-dimensional versions of these results are given in [8, 15]. Despite the apparent similarity between the definitions of PPAD and PPA, there is more progress in basing the hardness of PPA on standard cryptographic assumptions: \textsc{Factoring} can be reduced to PPA (with a randomised reduction) [26], while so far, the hardness of PPAD has relied on problems from indistinguishability obfuscation [6, 21]; Garg et al. [22] make progress in weakening the cryptographic assumptions on which to base the hardness of PPAD, but these are still less satisfying than in the case of PPA.

Examples of problems known to be PPA-complete include the following. Aisenberg et al. [1] introduce the problem \textsc{2D-Tucker}: suppose we have a colouring of an exponentially-fine grid on a square region, the colouring being concisely represented via a circuit. Tucker’s Lemma (the discrete version of Borsuk-Ulam) guarantees that if certain boundary conditions are obeyed, then two adjacent squares in the grid will get opposite colours. \textsc{2D-Tucker} is the search for such a solution, or alternatively a violation of the boundary conditions. As it happens, we use \textsc{2D-Tucker} as the starting-point for our reductions here. Deng et al. [15] show PPA-completeness for finding fully-coloured points of triangulations of various non-oriented surfaces; the colourings are presented concisely via a circuit. Recently, Deng et al. [16] showed that \textsc{Octahedral Tucker} is PPA-complete, reducing from \textsc{2D-Tucker} and using a snake-embedding style technique that packages-up the exponential grid in 2 dimensions, into a grid of constant size in high dimension. Beblov et al. [5] show PPA-completeness for novel problems presented in terms of arithmetic circuits representing instances of the Chevalley-Warning Theorem, and Alon’s Combinatorial Nullstellensatz. There remain other problems in PPA that are not defined in terms of circuits, and are conjectured to be PPA-complete. They include \textsc{Smith}, the problem of finding a second Hamiltonian cycle (given one as part of the input) in an odd-degree graph [34, 41], and the discrete Ham Sandwich problem [34] (given \(n\) sets of \(2n\) points in general position in \(n\)-space, find a hyperplane that splits each of these sets into two subsets of size \(n\)). Also the problem \textsc{Necklace-splitting} [2, 3], discussed in [34], Simmons and Su [40] note the connection with consensus-halving.

A precursor of this paper [19] established that the \textsc{Consensus-halving} problem is PPAD-hard, even when we allow constant-size approximation errors for the agents. Taken with the computational equivalence of Consensus-halving and Necklace-splitting established here, we immediately obtain PPAD-hardness of Necklace-splitting, thus in a well-established sense, Necklace-splitting is computationally intractable. This partially answers a question posed in [1] about the hardness of Necklace-splitting.

### 1.2 Overview of the Proof

We begin by explaining the ground covered by [19] (where PPAD-hardness was established), and then give an overview of the proof in the present paper. In [19], each agent \(a\) in a \textsc{Consensus-halving} instance, has a particular cut \(c(a)\) associated with \(a\). In an instance \(\mathcal{L}_CH\) of \textsc{Consensus-halving}, we refer to the interval \(A\) on which agents have valuation functions, as the \textit{domain} of \(\mathcal{L}_CH\).

[19] established PPAD-hardness by reduction from the PPAD-complete problem \(\varepsilon\text{-Gcircuit} (\varepsilon\text{-approximate Generalised Circuit})\) in which the challenge is to find a fixpoint of a circuit in which each node computes (with error at most \(\varepsilon\)) a real value in the range \([0, 1]\), consisting of a function of at most two other nodes in the circuit; these may be certain simple arithmetic operations, or boolean operations (regarding 0 and 1 as representing false and true respectively). In [19]’s reduction from \(\varepsilon\text{-Gcircuit} \) to \textsc{Consensus-halving}, each node \(v\) of a generalised circuit has a corresponding agent \(a_v\), and the value computed at \(v\) is represented by the position taken by the cut \(c(a_v)\). \(a_v\)’s valuation function is designed to enforce the relationship that \(v\)’s value has with the node(s) providing input to \(v\). Here we re-use some of the circuit “gate gadgets” of [19], in particular the boolean ones. A cut that encodes the value computed at a boolean gate is expected to lie in one of two short intervals, associated with true and false.

In moving from PPAD-hardness to PPA-hardness, we encounter a fundamental limitation to the above approach, which is that distinct cuts are constrained to lie in distinct (non-overlapping) regions of \(A\), and collectively, the cuts lie in an oriented domain. A new idea is needed, and we construct two special agents (the “coordinate-encoding agents”) along with two cuts that correspond to those agents, which are less constrained regarding where, in principle, they may occur, in a solution to the resulting \textsc{Consensus-halving} instance \(\mathcal{L}_CH\). These two cuts are regarded as representing a point on a Möbius strip, and a distance metric between two pairs of positions for these cuts, does indeed correspond to distance between points on a Möbius strip. New problems arise from this freedom regarding where these cuts can occur, mainly the possibility that one of them may occur outside of the intended “coordinate-encoding region” of the domain of \(\mathcal{L}_CH\). Consequently it may interfere with the circuitry that \(\mathcal{L}_CH\) uses to encode an instance of \textsc{2D-Tucker} (which, recall, is the problem we reduce from). We deal with this possibility by making multiple copies of the circuit, so that an unreliable copy is “out-voted” by the reliable ones. The duplication (we use 100 copies) of the circuit serves a further purpose reminiscent of the the “averaging manoeuvre” introduced in [14]: we need to deal with the possibility of values occurring at nodes of the circuit that fail to correspond to boolean values. The duplication corresponds to a sampling of a cluster of points on the Möbius strip, most of which get converted to boolean values.

One other significant obstacle addressed here, is due to the coordinate-encoding cuts directly representing the location of a point on the Möbius strip with exponential precision. We construct a novel mechanism that reads off \(\Theta(n)\) bits of precision from the locations of these cuts, which are then fed in to the circuit-encoding part of the consensus-halving instance. (It is this part of the proof that requires us to work with a definition of \(\varepsilon\text{-Consensus-halving} \)
that may require $\epsilon$ to be inverse exponential. PPA-hardness for in-
verse polynomial $\epsilon$ would lead to PPA-completeness of Necklace-
splitting, but there seems to be no way to achieve this while reduc-
ting from 2D-Tucker in a way that directly encodes the loca-
tion of a solution to 2D-Tucker.)

Our reductions start out from the 2D-Tucker result of [1]. In
Section 3 we give a straightforward proof of PPA-completeness of
a restricted version called 2D-MS-Tucker, in which two opposite
sides of the domain are each monochromatic. We look into this to
an artificial-looking problem called Variant Tucker, which is
effectively a messy-looking version of 2D-MS-Tucker: the purpose
of introducing Variant Tucker is to extract some of the technical clut-
ter from the main event, which is the reduction from there to
Consensus-Halving (Section 4).

2 PRELIMINARIES

2.1 The Consensus Halving Problem

Simmons and Su [40] were not concerned with computational is-
ues; their result is essentially topological and shows that a solution
exists provided that agents’ valuations are infinitely divisible. A
computational analogue requires us to identify how functions are
represented, and we assume they are given as step functions, or
piecewise constant functions, as have also been considered in the
cake-cutting literature [12, 35]. A problem instance also includes an
approximation parameter $\epsilon$, the allowed difference in value
between the two sides of the partition, applicable to any agent.

Definition 2.1. $\epsilon$-Consensus-Halving: An instance $I_{CH}$ incorpo-
rates, for each of $i \in [n]$, a non-negative measure $\mu_i$ of a finite line
interval $A = [0, x]$, where each $\mu_i$ integrates to 1 and $x > 0$ is part
of the input. We assume that $\mu_i$ are step functions represented in
a standard way, in terms of the endpoints of intervals where $\mu_i$ is
constant, and the value taken in each such interval. We use the bit
model (logarithmic cost model) of numbers. $I_{CH}$ also incorporates
$\epsilon \geq 0$ also represented using the bit model. We regard $\mu_i$ 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 $i$, $|\mu_i(A_+) - \mu_i(A_-)| \leq \epsilon$, i.e. each agent has
a value in the range $[\frac{1}{2} - \frac{\epsilon}{2}, \frac{1}{2} + \frac{\epsilon}{2}]$ for the subintervals labelled $A_+$
(thus, also values the subintervals labelled $A_-$ in that range).

A version where the domain $A$ is take to be $[0, 1]$ is polynomial
time equivalent to that of Definition 2.1 (by scaling the valuations
appropriately). In the instances that we construct, $x$ is polynomial
in $n$ but one can equivalently allow $x$ to be exponential in $n$; the
rescaling changes the size of the encoding of the problem instances
by a polynomial factor.

Note that it’s not hard to check that an instance of Consensus-
Halving is well-formed in the sense that the valuation functions
should integrate to 1. Also, note that the bit complexity of numbers
involved in an approximate solution need not be excessive, so,
together with the proof of [40] we have containment in PPA. A
couple of relevant remarks are the following:

- Definition 2.1 allows the accuracy parameter $\epsilon$ to be inverse
  exponential in $n$, which will be essential for our reduction. In
  fact, [19] established PPAD-hardness of the problem even for
  constant $\epsilon$. An interesting open question is whether our PPA-
  hardness result can be extended for constant or even inverse
  polynomial $\epsilon$ (which would lead to PPA-completeness for
  Necklace-splitting; Section 6).

- The fact that the functions $\mu_i$ are step-functions which inte-
grate to 1 over the whole interval $A$ is desirable, since this
makes the hardness result stronger, compared to arbitrary functions.
Note that while the step functions $\mu_i$ must have polynomially-many steps, the values they may take can differ
by exponential (in $n$) ratios. The “in PPA” result on the other
hand is established for arbitrary (bounded, non-atomic) functions,
which also makes it as strong as possible, given that it is a containment result.

Solutions with alternating labels: We assume without loss of gen-
erality that we seek solutions to Consensus-Halving in which the
labels $A_+$ and $A_-$ alternate as we consider the subintervals formed
by the cuts, from left to right. If, say, there are two consecutive
subintervals labelled $A_+$ in a solution, we could combine them into
a single subinterval, leaving us with a un-needed cut, which could
be placed at the right-hand endpoint of $A$. We can also assume
without loss of generality that the labelling sequence starts with
$A_+$ on the leftmost subinterval of $A$ defined by the set of cuts.

2.2 The 2D-Tucker Problem

We review the total search problem 2D-Tucker, as defined and
shown PPA-complete in [1], (Definition 2.2 is a variant of it, that
we use, that’s easily seen to be equivalent to the version of [1].)

An instance of 2D-Tucker consists of a labelling $\lambda : [m] \times [m] \rightarrow
\{\pm 1, \pm 2\}$ satisfying the boundary conditions: 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 $(x_1, y_1), (x_2, y_2)$
$(x_1, x_2, y_1, y_2 \in \{\pm 1\}^m)$ with $|x_1 - x_2| \leq 1$ and $|y_1 - y_2| \leq 1$ such that
$\lambda(x_1, y_1) = -\lambda(x_2, y_2)$.

In the above definition, $m$ is exponential, and $\lambda$ is presented via
a circuit that computes it. We use Definition 2.2, a variant of the above
whose PPA-completeness easily follows; it is a more convenient
version for us to use.

Definition 2.2. An instance $I_T$ of 2D-Tucker (with complexity
parameter $n$) is defined as follows. Consider the square region
$[0, 2^n] \times [0, 2^n]$. For $1 \leq i, j \leq 2^n$, the $(i,j)$-squarelet des-
otes the unit square whose top right vertex is at $(i,j)$. $I_T$ consists of a
boolean circuit $C$ having $2n$ input bits representing the coordinates
of a squarelet, and 2 output bits representing values 1, -1, 2, -2. $C$’s
labelling should obey the boundary conditions of [1] noted above.
A solution consists of two squarelets that touch at at least one point,
and have opposite labels (i.e. labels that sum to 0).

The containment of the problem in PPA was known from [34].
Aisenberg et al. [1] proved that the problem is also PPA-hard.

Theorem 2.3. $[1, 34]$ 2D-Tucker is PPA-complete.
2.3 Organization of the Paper

In Section 3, we reduce from 2D-Tucker (the version of Definition 2.2) to a restricted version 2D-MS-Tucker where two opposite sides of the Tucker square are completely labelled 1 and -1 (a monochromatic sides version). From there, we reduce to an artificial looking variant, Variant Tucker, which however will prove to be very useful for our main reduction to Consensus-Halving. Section 4 presents the main reduction and in Section 5, we establish the correctness of the reduction. Finally, in Section 6, we show a computational equivalence between approximate Consensus Halving and the well-known Necklace Splitting problem. Due to lack of space, we omit some of the details, which are included in the full version of the paper.

3 REDUCING FROM 2D-TUCKER TO VARIANT-TUCKER

In this section, we reduce from 2D-Tucker to a variant of the Tucker problem, which will be more appropriate to use for proving PPA-hardness of approximate Consensus Halving. The PPA-hardness of the Variant Tucker problem will be established through a sequence of two reductions.

First, we reduce from 2D-Tucker to a version of the problem when two opposite sides of the square are assigned only a single label (with opposite signs), e.g., 1 and -1 (Definition 3.1). We will refer to this problem as the 2D-MS-Tucker (where MS stands for "monochromatic sides"). See Definition 3.1.

Definition 3.1. An instance $I_{MS}$ of 2D-MS-Tucker (with complexity parameter $n$) is defined similarly to an instance $I_T$ of 2D-Tucker but in addition, all squarelets $(x,y)$ with $y=1$ get labelled 1, and all squarelets $(x,y)$ with $y=2^{n}$ get labelled -1. (So, two opposite sides are monochromatic.) As before, a solution consists of two squarelets that touch at least one point.

We start from the PPA-hardness of 2D-MS-Tucker. The proof of the following lemma is conceptually simple, and we leave it for the full version.

Lemma 3.2. 2D-Tucker is polynomial-time reducible to 2D-MS-Tucker.

Then, we reduce from 2D-MS-Tucker to Variant Tucker, by embedding the regions of the 2D-MS-Tucker instance (i.e., the squarelets) into a triangle-domain and extending the labelling function to points outside these regions. In this process, there is a designated significant sub-domain which contains the embedded regions along with diagonal strips that emerge from the embedded regions and go out the edge of the triangle-domain. The embedding is such that the lines separating the regions are piecewise rectilinear, with sufficiently long pieces. Intuitively, the regions will not be separated by diagonal lines but rather by "zig-zag" rectilinear lines that approximate the diagonal ones, which results in set of regions that we refer to as tiles. See Figure 1.

Variant Tucker is essentially a technically-cluttered version of 2D-MS-Tucker. It’s helpful as an intermediate stage towards our eventual goal of Consensus-Halving, since the technical clutter emanates from the way we encode 2D-Tucker in terms of Consensus-Halving, and by reducing from Variant Tucker, we simplify the proof that our final reduction to Consensus-Halving does indeed work.

Definition 3.3. A subregion of the plane consists of an equivalence class of points $(x,y)$ that are equivalent when their binary expansions are truncated after $n+4$ bits of precision; thus any subregion is a square with an edge length of $\frac{1}{16}2^{-n}$.

Definition 3.4. Consider pairs $(a,b)$ of non-negative even numbers, for which either $a$ and $b$ are both multiples of 4, or neither are.

Define the $(a,b)$-tile to be a union of 8 subregions arranged as in Figure 1, with central point at $\left(\frac{1}{4}a \cdot 2^{-n}, \frac{1}{16}b \cdot 2^{-n}\right)$, having a height and width of $\frac{1}{16}2^{-n}$. (Thus all horizontal and vertical line segments have coordinates that are multiples of $\frac{1}{16}2^{-n}$.) If $a$ or $b$ is equal to zero, the tile consists of just the parts of this region with non-negative coordinates.

Observe that (for the values of $a,b$ allowed in Definition 3.4) tiles tessellate the positive quadrant of the plane as in Figure 1.

Definition 3.5. An instance of Variant Tucker with complexity parameter $n$, consists of a boolean circuit $C$ that takes as input $2n+22$ bits. These input bits represent the coordinates of a point $(x,y)$ for $x,y \in [0,1]$, each of $x$ and $y$ represented as a bit string with $n+11$ binary places of precision. $C$ has 4 boolean outputs that we use to represent the values $1, -1, 2, -2$, respectively as 1110, 0001, 0111, 1000. $C$ obeys the following constraints that may be enforced syntactically:

1. If $y < x - \frac{1}{3}$ then $C$ must output 1;
2. If $y > x - \frac{1}{3}$ then $C$ must output -1;
3. If $y > x + \frac{1}{3}$ then the output of $C$ should be opposite to its output on $(1-x, 1-y)$, and similarly for points with $y < x - \frac{1}{3}$;
4. The output of $C$ may not depend on the last 7 bits of $x$ or $y$;
5. Moreover, the output value of $C$ is constant within tiles (Definition 3.4, Figure 1): a tile consists of 8 square regions with 128 discrete points along their edges, arranged as in Figure 1.

6. We allow the following exception to the above rules, which is that for input bit-strings that represent points that lie adjacent to the boundary of any subregion, $C$’s output value is unrestricted.

A solution consists of a sequence of 100 points $(x_i, y_i)$ for $1 \leq i \leq 100$, where $y_1 \leq x_1$, and for $i > 1$ we have $x_i = x_{i-1} + 2^{-(n+1)}$ and $y_i = y_{i-1} - 2^{-(n+1)}$, where addition and subtraction are taken modulo 1. These 100 points should contain a set of 10 points that all produce the same output, and another set of 10 points that produce the opposite output. In the case that $y_1 < 100 \cdot 2^{-(n+1)}$ and the sequence of points "wraps around", this property must instead hold after we negate the outputs of the wrapped-around subsequence.

Lemma 3.6. Variant Tucker is PPA-complete.

Proof. We reduce from 2D-MS-Tucker. Squarelets in an instance $I_{2DMST}$ of 2D-MS-Tucker correspond to tiles in an instance $I_{MS}$.
I_{VT}$ of Variant Tucker as follows. $I_{DMST}$ contains squarelets $(i,j)$ for $1 \leq i,j \leq 2^n$. Each $(i,j)$ squarelet determines the value taken by $C$ on the $(2 \cdot 2^n + 2i + 2j, 4 \cdot 2^n - 2i + 2j)$-tile. With this rule, the squarelets of $I_{DMST}$ are mapped into tiles in the region $R$ in Figure 2 in such a way that adjacencies are preserved: two squarelets are adjacent if and only if their corresponding tiles are adjacent.

Suppose that the monochromatic sides of $I_{DMST}$ are squarelets $(i,j)$ with $j = 1$ having label 1, and squarelets $(i,j)$ with $j = 2^n$ having label $-1$. As a result, these squarelets get mapped to sides of the region $R$ that are adjacent to and match the monochromatic regions adjacent to $R$ (the regions where $y < \frac{y}{2} - x$, alternatively $y > \frac{y}{2} - x$). Any tile in the remaining parts $R'$, $R''$ of the triangular domain of Figure 2 is allocated the same colour as its closest (Euclidean distance) tile in $R$. That is, the colour of the $(2 \cdot 2^n + 2j, 4 \cdot 2^n + 2j)$-tile is allocated to the $(2 \cdot 2^n + 2j - 2k, 4 \cdot 2^n + 2j + 2k)$-tile, for positive integers $k$, and the colour of the $(4 \cdot 2^n + 2j, 2 \cdot 2^n + 2j)$-tile is allocated to the $(4 \cdot 2^n + 2j + 2k, 2 \cdot 2^n + 2j + 2k)$-tile, for positive integers $k$. Notice that this rule obeys Property 3 of Definition 3.5, due to the boundary condition on the colouring of squarelets in $I_{DMST}$.

Given that $I_{DMST}$ has a concise circuit that labels its squarelets, it’s not hard to see that the corresponding instance $I_{VT}$ has a concise circuit that takes as input, points in the triangular region (at the slightly higher numerical precision), checks which tile a point belongs to, and labels it according to the above rules. We claim that for a sequence of 100 points to contain two sets of 10 points having opposite labels, as required for a solution, this will only happen when that sequence crosses two adjacent tiles having opposite labels. Any sequence of 100 points constructed as in the problem definition, may cross the boundaries of subregions in at most 2 places, resulting in at most 4 points where $C$ can disobey the tile colouring due to the exception in item (6). So most of the points in the two sets of 10 oppositely-labelled points must indeed come from two oppositely-labelled tiles.

If this happens in region $R$ of Figure 2, the two tiles correspond directly to two adjacent squarelets in $I_{DMST}$ having opposite labels.

It could also occur in the regions $R'$ or $R''$, in which case we find a solution in the closest edge of $R$. Suppose the sequence of 100 points “wraps around”, i.e. straddles $R'$ and $R''$, crossing the line between $(0,0)$ and $(1,0)$ and appearing just to the right of the line between $(0,0)$ and $(0,1)$. Labels get negated at the point where we wrap around, but recall that in this case, we flip the suffix of the sequence occurring in $R'$, before applying the test that two sets of 10 points have equal and opposite labels. From such a sequence of points we can identify either of two solutions on the north-west or south-east sides of $R$ that are closest to the sequence of points. □

4 THE CONSENSUS HALVING INSTANCE

In this section, we describe how to construct an instance $I_{CH}$ of $\epsilon$-Consensus-halving from an instance of Variant Tucker, for inverse-exponential precision parameter $\epsilon$. At a high level, the domain $A$ of $I_{CH}$ will have two designated regions — a small one, typically containing 2 cuts in a solution, which represent coordinates of points in the triangular domain of Figure 2 (the “coordinate-encoding region”) and a larger one for the encoding of the labelling circuit (the “circuit-encoding region”). Certain sensing gadgets will detect the position of coordinate-encoding cuts and will feed this information to a set of gadgets which encode the inputs to the labelling circuit of the Variant Tucker instance. This information will be propagated through the circuit-encoding gadgets and fed back to the coordinate-encoding region. The idea is that two designated agents of the Consensus Halving instance, which will be associated with the coordinate-encoding region, will only be satisfied with the balance between $A_+$ and $A_-$. If the detected cuts correspond to points on a sequence that is a solution to Variant Tucker (Sections 4.3 and 4.4).

The construction will actually encode multiple copies of the labelling circuit of Variant Tucker for two different reasons. The main reason is that for each copy of the circuit, the cuts in the coordinate-encoding region will encode a different point in the domain, with these points being sufficiently close to each other (this will be achieved by small shifts in the valuation blocks corresponding to the circuits in the coordinate-encoding region) and with all of them lying on the same line segment. We will ensure that a solution to $I_{CH}$ will correspond to (sufficiently many) points of this segment with coordinates in squarelets with equal and opposite labels. The other reason is to deal with “stray cuts”, i.e. cuts that are
intended to lie in the coordinate-encoding region but actually cut through the circuit-encoding region. These cuts might “invalidate” the circuits that they cut through, but the construction will ensure that the rest of the circuits will remain unaffected, and there will still be sufficiently many reliable points in the sequence (Section 4.5 and Section 5).

More concretely, given an instance $I_{VT}$ of Variant Tucker with complexity parameter $n$, we will construct an instance $I_{CH}$ of $\epsilon$-Consensus-Halving for $\epsilon = 2^{-2n}$. Let $A$ denote the Consensus-Halving domain, an interval of the form $[0, x]$ where $x$ is of size polynomial in $n$. Any agent $a$ in $I_{CH}$ has a measure $\mu_a : A \to \mathbb{R}$ which will be represented by a step function (having a polynomial in $n$ number of steps).

### 4.1 Regions and Agents of the Instance $I_{CH}$

The domain $A$ will consist of two main regions:
- The coordinate-encoding region $[0, 1]$ (or simply $c-e$ region).
- The circuit-encoding region $(1, x]$ (abbreviated as $R$).

Our construction contains 100 copies of an encoding of the labelling circuit $C$ of Variant Tucker and for the purpose, the circuit-encoding region $R$ will be further divided into 100 non-intersecting sub-regions $R_1, \ldots, R_{100}$, one for each copy of the circuit. The regions $R_i$ are of equal length and constitute a partition of $R$. We further divide each region $R_i$ into three sub-regions $R_{i1}, R_{i\text{mid}}$ and $R_{i2}$, which again are non-intersecting and partition $R_i$. These regions correspond with parts of a circuit that deal respectively with the input bits, the intermediate bits and the outputs.

The instance $I_{CH}$ will have the following sets of agents:

2 coordinate-encoding agents $a_1, a_2$ whose valuation functions $\mu_{a_1}, \mu_{a_2}$ are only positive in $\bigcup_{i=1}^{100} R_i^{\text{out}}$ (See Subsection 4.3).

100 circuit-encoders $C_1, \ldots, C_{100}$ (see Subsection 4.4).

- Each $C_i$ has an associated circuit-encoding region $R_i$.
- With each $R_i$, there is a polynomial number of associated circuit-encoding agents. Let $A_i$ be the set of those agents; the set $A_i$ consists of the following sets of agents:
  - A set $S_i \subseteq A_i$ of $8(n + 8) + 1$ sensor agents with value in $[0, 1] \cup R_i^{\text{in}}$. Among those, there will be a designated agent that we will refer to as the blanket sensor agent. (See Subsection 4.4.1).
  - A set $G_i \subseteq A_i$ of polynomially-many gate agents, with value in $R_i^{\text{mid}} \cup R_i^{\text{out}}$. (See Subsection 4.4.2).

We associate one cut with each agent; recall that for agent $a$, $c(a)$ is the cut associated with the agent. In a solution to $I_{CH}$, for any agent $a_i \in A_i$, these cuts will lie in a specific region, where most of the value of agent $a_i$ will be concentrated. We will use $R(a_i)$ to denote this region. The cuts $c(a_1), c(a_2)$ for the coordinate-encoding agents, are called the coordinate-encoding cuts and the associated region for them is the $c-e$ region, i.e. $R(a_1) = R(a_2) = [0, 1]$. We will see that in any solution, either both or one of the coordinate-encoding cuts must lie in the coordinate-encoding region and the other cuts must lie in region $R$. In the event that a coordinate-encoding 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, we will see that the duplication of the circuit using

![Figure 3: 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.](image)
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.

Boolean gate gadgets: Consider a boolean gate that is an AND, an OR, or a NOT gate, denoted $g_a$, $g_v$, and $g_r$, respectively. Let $i_{in1}$, $i_{in2}$, and $o_{out}$ be intervals such that $|i_{in1}| = |i_{in2}| = |o_{out}| = 1$. We will encode these gates using the following gate-gadgets.

$$g_{\cdot}(i_{in1}, o_{out}) = \begin{cases} 0.25 & \text{if } t \in i_{in1} \\ 7.5 & \text{if } t \in [r(o_{out}) - 1/20, r(o_{out})] \\ 0 & \text{otherwise} \end{cases}$$

$$g_{\cdot}(i_{in2}, o_{out}) = \begin{cases} 0.125 & \text{if } t \in i_{in1} \cup i_{in2} \\ 6.25 & \text{if } t \in [r(o_{out}) - 1/20, r(o_{out})] \\ 0 & \text{otherwise} \end{cases}$$

$$g_{\cdot}(i_{in1}, i_{in2}, o_{out}) = \begin{cases} 8.75 & \text{if } t \in [r(o_{out}) - 1/20, r(o_{out})] \\ 0 & \text{otherwise} \end{cases}$$

Note that the gadget corresponding to the NOT gate only has one input, whereas the gadgets for the AND and OR gates have two inputs. In the interval $o_{out}$, 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 on the right hand side is $A_+$. This can be achieved with the appropriate use of parity gadgets.

**Observation 1.** The boolean gate gadgets described above encode valid boolean NOT, OR, and AND operations. The proof of the statement follows rather easily from the definitions of the gadgets and is left for the full version.

### 4.3 The Coordinate-Encoding Agents

The coordinate-encoding region $[0, 1]$ is the region from which the value of the solution to $\epsilon$-Consensus-Halving will be read and will be translated to coordinates of a grid point on $I_{1:T}$. Associated with this region, there are 2 coordinate-encoding agents $a_1$ and $a_2$. However, these agents will have 0 value in the subinterval $[0, 1]$ and all of their value will lie in the circuit-encoding region $R_i$ and specifically in $\cup_{i=1}^{100} R_i^{out}$.

The value of the c-e agents in $R_i^{out}$ will correspond to the feedback mechanism from the blanket sensor agent of $R_i$ (see Subsection 4.4.1) and the feedback mechanisms from the gate-agents corresponding to the output gates of the circuit (see Subsection 4.6). Concretely the valuation of the coordinate-encoding agents is defined as follows:

$$\mu_{\cdot}(t) = \begin{cases} 30/800 & \text{if } t \in \cup_{i=1}^{100} [f(R_i^{pre}) + 1/10, f(R_i^{out}) + 2/10], \\ 30/800 & \text{if } t \in \cup_{i=1}^{100} [f(R_i^{out}) + 1 - 2/10, f(R_i^{out}) + 9/10], \\ 1/400 & \text{if } t \in \cup_{i=1}^{100} [f(R_i^{out}) + 1.25, f(R_i^{out}) + 1.75], \\ 1/400 & \text{if } t \in \cup_{i=1}^{100} [f(R_i^{out}) + 3.25, f(R_i^{out}) + 3.75]. \end{cases}$$

Note that for each $i = 1, \ldots, 100$, the value of the agent in $R_i$ adds up to 1/100 and therefore the agent’s total valuation in $R$ is 1. Intuitively, each coordinate encoding region has values that consist of the following components in each region $R_i$:

- A bit detection gadget positioned in the interior of the interval where the bit detection gadget of the corresponding blanket sensor agent is situated (see also Subsection 4.4.1).
- Two blocks of valuation situated in the interior of the intervals where the bit detection gadgets of the output gate agents are situated in $R_i^{out}$ (see also Subsection 4.4.2). There are four output gate agents in each $R_i$; $a_1$ has value in the corresponding intervals for two of those and $a_2$ has value in the corresponding intervals for the other two.

### 4.4 The Circuit Encoders

In this subsection, we explain how to design the circuit-encoders $C_1, \ldots, C_{100}$. Recall that these are sets of agents of $I_{CH}$ that encode the labelling circuit $C$ of Variant Tucker, including the inputs and the outputs to the circuit, via the use of sets $A_1, \ldots, A_{100}$ of circuit-encoding agents. In the set $A_i$, there are two different types of circuit-encoding agents:

- **The sensor agents $S_i$** that are responsible for extracting the binary representation of the positions of the cuts in the c-e region, which will be used as inputs to the remaining circuit-encoding agents. These agents have value in $[0, 1] \cup R_i^{out}$. Among those, there is a designated agent $a_i^{bt}$ that we refer to as the blanket sensor agent.

- **The gate agents $G_i$** that implement a circuit $C_i$, consisting of subcircuits: a pre-processing circuit $C_i^{pre}$ and a main circuit $C_i^{main}$, which further consists of a copy $C_i^{out}$ of the labelling circuit $C$ of Variant Tucker as well as a small “XOR operator” circuit $C_i^{det}$. The pre-processing circuit will be responsible for transforming the information extracted from the sensor agents into the encoding of a point on the domain, which is then fed to $C_i^{out}$. In particular, for each gate of the circuit $C_i$, we will have one associated agent of $I_{CH}$. For each gate agent that corresponds to an input gate of $C_i$, the agent will have value in $R_i^{pre} \cup R_i^{mid}$ and for each gate agent that corresponds to an output gate of $C$, the agent will have value in $R_i^{out} \cup R_i^{mid}$. All other gate agents will have value only in $R_i^{mid}$. (See Subsection 4.4.2.) In the next subsections, we design the values of those agents explicitly. We will first explain how to construct the circuit-encoder $C_1$ and then based on this, we will construct the remaining circuit-encoders $C_2, \ldots, C_{100}$.

#### 4.4.1 The Sensor Agents

In this subsection, we will design the set of sensor agents, which is perhaps the most vital part of the construction. Roughly speaking, these agents will be responsible for detecting the position of a cut in the c-e region and extracting its binary representation. The set $S_1$ contains $8(n + 8) + 1$ sensor agents, which consist of...
the blanket sensor agent $a^{bs}_i$, that is responsible for detecting large discrepancies in the lengths of $A_s$ and $A_e$ in the c-e region, $8(n + 8)$ bit-extractors: 8 sets of $n + 8$ agents, each set responsible for extracting a bit string of length $n + 8$, which indicate the positions of cuts with respect to 8 different intervals that span the c-e region; we will refer to these inputs as the raw data. The raw data will then be inputted by the pre-processing circuit-encoding $C^{pre}_i$ and will be transformed into the $n + 11$ most significant bits of the binary representation of the positions of the cuts.

The Blanket Sensor Agent. The valuation of the blanket sensor agent $a^{bs}_i$ is defined as:

$$
\mu_{a^{bs}_i}(t) = \begin{cases} 
0.1 & \text{if } t \in [0, 1], \\
8.5 & \text{if } t \in [(\ell(R_1^{out}) + 1/20, \ell(R_1^{out}) + 1), \\
8.5 & \text{if } t \in [(\ell(R_1^{out}) + 19/20, \ell(R_1^{out}) + 1), \\
0.1 & \text{if } t \in [(\ell(R_1^{out}) + 1/4, \ell(R_1^{out}) + 3/4).
\end{cases}
$$

In other words, the blanket sensor agent has a valuation block of volume 0.1 spanning over the whole coordinate encoding region and region and two dense valuation blocks of volume 0.425 over the intervals $[(\ell(R_1^{out}), \ell(R_1^{out}) + 1/20]$ and $[(\ell(R_1^{out}) + 19/20, \ell(R_1^{out}) + 1), with a valuation block of volume 0.05 between them, in $[\ell(R_1^{out}) + 0.25, \ell(R_1^{out}) + 0.75].$ Note that this latter part of the quasi is similar to a bit detection gadget, except for the fact that there is a small valuation block in between the two valuation blocks of large volume, which still constitute most of the agent’s valuation over $A$. Furthermore, note that since the length of $R_1^{out}$ is polynomial in $n$, the whole valuation of agent $a^{bs}_i$ lies in $[0, 1] \cup R_1^{out}$. The blanket sensor agent is responsible for detecting large enough discrepancies in $A_s$ and $A_e$. As we will see, if such a discrepancy exists, the blanket sensor agents will provide feedback to the c-e agents, making sure that this is not a solution to $I_{CH}$. We state the lemma here but we leave the proof for the full version.

Lemma 4.1. Let $A^{c-e}_s$ and $A^{c-e}_e$ be the total fraction of the c-e region labelled $A_s$ and $A_e$, respectively. The blanket sensor agents ensure that in a solution to $I_{CH}$, it holds that $|A^{c-e}_s - A^{c-e}_e| \leq 1/4$.

Whenever the blanket sensor agent $a^{bs}_i$ does detect such a discrepancy (and therefore the cut $c(a^{bs}_i)$ in $R(a^{bs}_i)$ assumes one of the extreme positions, left or right), we will say that the blanket sensor agent is active and that it overrides the circuit $C_i$. Otherwise, we will say that the blanket sensor agent is passive.

The Bit Extractors. The second set of agents in $S_1$ will be responsible for detecting the position of the cuts and extracting their binary expansion. To be more precise, these agents will extract 8 binary numbers of length $n + 8$, from 8 consecutive intervals of length $1/8$ each, which span the c-e region, and this number will encode the position of the cut within the interval. We will refer to these extracted binary strings as the raw data.

Lemma 4.1 ensures that it is not possible for two cuts to intersect the same interval of length $1/8$. If for some interval of length $1/8$ there are no cuts intersecting it, the corresponding bit extractors will output a binary string which consists of only 1’s or only 0’s; we will refer to such bit strings as solid.

Definition 4.2 (Solid String). A binary string is called solid if either all of its bits are 1 or all of its bits are 0.

If the interval is intersected by one cut, the bit extractors will output a binary string consisting of a non-trivial mixture of 0’s and 1’s.

The raw data extracted from the bit-extractors will be fed into the encoders of the input gates of $C_i$, and in particular to the pre-processing circuit $C^{pre}_i$ that will transform the extracted information into the binary representation of the coordinate of a point $(x, y)$ on the domain, which will then be fed into the encoding $C^{out}_i$ of the labelling circuit $C$. We explain this in more detail in Section 4.4.2.

For each bit extractor, there are another $n + 7$ sensor agents that will have exactly the same value in $[0, 1]$. In particular, this value will be 1/10 in volume, spanning over an interval of length $1/8$. We will refer to these $n + 8$ sensor agents as c-e identical, precisely because they have the same valuation in the c-e region. There will be exactly 8 sets of $n + 8$ c-e identical sensor agents. For $i = 2, \ldots, 8$, the values of the c-e identical agents for $i$ will be shifted by 1/8 to the right, compared to the values of the c-e identical agents of $i - 1$.

Therefore, the set of sensor agents covers the whole c-e region.

We will use $a^{s}_{i,k}$ to denote a bit extractor agent, where $j \in \{1, \ldots, 8\}$ and $k \in \{1, \ldots, n + 8\}$. Note that here, we drop the subscript referring to the specific circuit encoder $C_i$ for ease of presentation and since there is no ambiguity.

The agents in $[0, 1/8]$: First, we will define the valuations of agents $a^{s}_{i,k} = a^{s}_{i,k} \ldots a^{s}_{i,n + 8}$ and we will explain how to construct the valuations of the remaining agents from these agents. Note that these agents are c-e identical. First, let

$$
\mu_{a^{s}_{i,k}}(t) = \begin{cases} 
4/5 & \text{if } t \in [0, 1/8] \\
9 & \text{if } t \in [(\ell(R_1^{in}) + 1/20, \ell(R_1^{in}) + 1] \\
9 & \text{if } t \in [(\ell(R_1^{in}) + 1/20, \ell(R_1^{in}) + 1].
\end{cases}
$$

Then, define for $k = 2, \ldots, n + 8$,

$$
\mu_{a^{s}_{i,k}}(t) = \begin{cases} 
4/5 & \text{if } t \in [0, 1/8] \\
9 & \text{if } t \in [(\ell(R_1^{in}) + 1/20, \ell(R_1^{in}) + 1] \\
9 & \text{if } t \in [(\ell(R_1^{in}) + 1/20, \ell(R_1^{in}) + 1].
\end{cases}
$$

where above $R_{c_2} = [\ell(R_1^{in}) + 2(k - 1), \ell(R_1^{in}) + 2(k - 1) + 1/20]$ and $R_{c_2} = [\ell(R_1^{in}) + 2(k - 1) + 1/20, \ell(R_1^{in}) + 2(k - 1) + 1/20].$ Additionally, for $j = 1, \ldots, k - 1$, for $t \in [(\ell(R_1^{in}) + 2j + 0.25, \ell(R_1^{in}) + 2j + 0.75],$ we have that

$$
\mu_{a^{s}_{i,k}}(t) = 1/5 \cdot (1/2)^j.
$$

We now have the following proposition, the proof of which we postpone until Section 5.

Proposition 4.3. Given a cut in the interval where $n + 8$ c-e identical bit extractors have their value in the c-e region (e.g. the interval $[0, 1/8]$ for the first $n + 8$ c-e identical agents), the bit extractors recover a binary string of length $n + 8$ which encodes the cut position in that interval.

The remaining bit extractors: Next, we design the remaining 8 sets of c-e identical agents. These will be shifted versions of the first $n + 8$ bit-extractors, where their valuations in the c-e region will be shifted by 1/8 to the right (thus spanning the whole c-e region) and their valuations in $R_1$ will lie in “clean”, non-overlapping intervals.
The proof of the proposition is left for Section 5.

4.4.2 The Gate Agents. In this section, we will design the agents that will be responsible for encoding (i) the pre-processing circuit $C_{\text{pre}}^i$ that transforms the raw data into coordinates of points $(x, y)$ of the domain and (ii) the circuit $C_{\text{main}}^i$, which will consist of the encoding $C_{\text{VT}}^i$ of the labelling circuit of Variant Tucker, as well as a "XOR operator" circuit that will flip the label of the final outcome when needed. These agents will eventually provide feedback (in terms of a discrepancy of labels $A_1$ and $A_k$) to the c-e agents.

We omit the details of the construction here and instead provide the high-level idea of the operation of these sub-circuits; the explicit construction of this set of agents can be found in the full version.

Implementing the Circuit Using the Gate Gadgets. For both circuits (which we will view as a combined circuit in our implementation), at a high level, we will simulate the gates by gate agents, using the boolean gate gadgetry that we presented in Subsection 4.2, in the most straightforward way. In particular, for any two-input gate $g$ of the circuit with inputs $i_{n1}$, $i_{n2}$, agent $a_\ell^g$ will have a bit detection value gadget that will encode the output of the gate and furthermore, it will have value in some intervals $R_1$ and $R_2$ where the values of $i_{n1}$ and $i_{n2}$ lie respectively, where $i_{n1}$ and $i_{n2}$ can either be the outputs of some gates $g_1$, $g_2$ of some previous level, or the outputs of the sensor agents, if $g$ is an input gate of $C_{\text{pre}}^i$. The case of $g$ being a single-input gate is similar. The construction will make sure that agent $a_\ell^g$ will only be satisfied with the consensus-halving solution if the gate constraint is satisfied.

Concretely, we will use the gate gadgets from Subsection 4.2 that will encode the gates of the circuit. For each gate of $C_i$, we will associate a gate agent $a_{11}^g, \ldots, a_{|C_i|}^g$ with valuation given by the gadget

$$
\mu_{a_\ell^g}(t) = \begin{cases} 
g_T(i_{n1}^1, out^1) & \text{if } T = \overline{\land} \\
g_T(i_{n1}^2, i_{n2}^2, out^2) & \text{if } T \in \{\lor, \land, \overline{\lor}, \overline{\land}\}
\end{cases}
$$

where $i_{n1}^1$, $i_{n2}^2$ and $out^2$ are non-overlapping intervals that will be defined separately based on whether $a_\ell^g$ corresponds to an input gate, an output gate or an intermediate gate of the circuit. Again here, we drop the subscript corresponding to the circuit-encoder $C_i$ for notational convenience.

The Pre-Processing Circuit $C_{\text{pre}}^i$. As we mentioned earlier, the pre-processing circuit inputs the raw data extracted from the circuit encoders and outputs the binary expansion of the coordinate of the detected position $(x, y)$ in the c-e region. Then, the outcome of $C_{\text{pre}}^i$ is fed directly into the input gates of $C_{\text{VT}}^i$ and the information is propagated through the circuit, resulting in the assignment of a label for the encoded point. The circuit can decide how to interpret the bit extracted from region $R_k$ based on the raw data read from the bit-encoders for regions $R_1, \ldots, R_{k-1}$, particularly whether this is a string of 1’s or a string of 0’s.

The Main Circuit $C_{\text{main}}^i$. The encoding of the circuit $C_{\text{main}}^i$ will consist of the encodings of two sub-circuits, the labelling circuit $C_{\text{VT}}^i$ of Variant Tucker and the XOR operator circuit. The input to the circuit $C_{\text{VT}}^i$ is the binary representations of the coordinates of a grid point within a squarelet of $I_{\text{VT}}$ outputted by the pre-processing circuit. Recall that each squarelet contains a set of grid points with a resolution of $2^7$ in each dimension (see Figure 1). The output is a label $(\pm 1, 2z)$; in particular, the output gates of $C$ are $g_1^\pm_{\text{out}}, g_1^+_{\text{out}}, g_1^-_{\text{out}}$ and $g_1^0_{\text{out}}$ and the corresponding correspondence is syntactically enforced (Definition 3.5):

$$(1 \to 1110), (\pm 1 \to 0001), (2 \to 0111), (\mp 2 \to 1000)$$
The virtual cuts: For the circuit encoders $C_2, \ldots, C_{100}$ it will often be useful to think of the following alternative interpretation of their inputs. Consider the two cuts (the case of one cut is similar) in the c-e region, at positions $c_1^1, c_2^1$, encoding a point $(x, y)$ of the domain (also see Section 4.5). Since $C_2$ is a version of $C_1$ where all the values in the c-e region are shifted by $2^{-(n+1)}$ to the left (wrapping around for some valuations), we can equivalently think of the output of $C_2$ as what the output of $C_1$ would have been if the cuts had been moved slightly to the right, i.e. to $c_1^2 = c_1 + 2^{-(n+1)}$ and $c_2^2 = c_2 + 2^{-(n+1)}$ respectively. In other words, for each circuit-encoder $C_i$, we can think of its output as the output of $C_1$ if the cut were placed at $c_1^i$ and $c_2^i$. We will refer to such cuts as virtual cuts.

### 4.5 Recovery of a Solution of $I_{VT}$ from a Solution of $I_{CH}$

In this subsection, we explain how to obtain a solution to $I_{VT}$ from a solution to $I_{CH}$. Recall from Section 3 that a solution to $I_{VT}$ is a sequence of points $(x_1, y_1), (x_2, y_2), \ldots, (x_{100}, y_{100})$ of the (discrete) domain, lying on a line segment such that two sets of at least 10 of these points each have coordinates in squarelets of equal and opposite labels. Specifically, for each point $(x_i, y_i)$ on the segment, with $i \leq 100$, it holds that $x_{i+1} = x_i + 2^{-(n+1)}$ and $y_{i+1} = y_i - 2^{-(n+1)}$ (See Figure 1). Now consider a solution $\mathcal{H}$ to $I_{CH}$. As we will establish in Section 5, in $\mathcal{H}$ there must exist one or two cuts situated in the coordinate-encoding region $[0, 1]$.

- If there is only one cut in $[0, 1]$, situated at $z \in [0, 1]$, let $x = 0$ and $y = 1 - z$ be the coordinates of a point on the domain.
- If there are 2 cuts in $[0, 1]$, situated at $z, z'$, let $x = z$ and $y = 1 - z'$ be the coordinates of a point in the domain.

If we use $n + 11$ bits of precision to represent the coordinates $(x, y)$ of the point that correspond to the solution $\mathcal{H}$ above, we end up with the closest point $p$ on the discrete domain to $(x, y)$. Then, we can obtain a solution to $I_{VT}$ by generating a sequence of points $p_1, p_2, \ldots, p_{100}$ by setting $p_1 = p$ and $p_i = (x_{i-1} + 2^{-(n+1)}, y_{i-1} - 2^{-(n+1)})$ for $2 \leq i \leq 100$.

### 4.6 The Feedback Mechanism to the c-e Agents

Now that we have explained how the construction of $I_{CH}$ looks like, we can explain how the c-e agents receive feedback from the circuit. The agents will only be satisfied with the partition of labels if the line segment of points $(x, y)$ crosses a boundary of tiles with same and opposite labels and there are sufficiently many points indisputably labelled with each one of those labels.

First note that for a point $(x_i, y_i)$ of the domain recovered as described in Section 4.5, each point $(x_i, y_i)$ in the sequence of 100 points will be labelled by a different copy of the circuit $C_i$. Consider such a copy and let $C_i(x_i, y_i)$ be its output; recall that $C_i(x_i, y_i) \in [1110, 0001, 0111, 1000]$ (which can be syntactically enforced) and furthermore, we have the following correspondence:

$$(1 \rightarrow 1110), \quad (2 \rightarrow 10111) \quad \text{and} \quad (2 \rightarrow 10000).$$

Assume that the $C_j(x_i, y_i) = j$ for some $j \in [1110, 0001, 0111, 1000]$ and let $\tilde{L}_j \in \{1, -1, 2, -2\}$ be the corresponding label. For each of the c-e agents $a_1$ and $a_2$, the contribution to $A_+$ from its valuation on $R_{\text{out}}^{i, \text{out}}$ is $2$ for $a_1$ and $-2$ respectively.\footnote{Assuming that $C_j$ behaves as expected, i.e. it receives good inputs and is reliable - see Section 5 for the definitions.}
\[
\begin{align*}
&\begin{cases}
    \frac{2}{100}, & \text{if } C_i(x_i, y_i) = 1110 \\
    \frac{2}{99}, & \text{if } C_i(x_i, y_i) = 0001 \\
    0, & \text{otherwise},
    \end{cases} \\
&\begin{cases}
    \frac{2}{100}, & \text{if } C_i(x_i, y_i) = 0111 \\
    \frac{2}{99}, & \text{if } C_i(x_i, y_i) = 1000 \\
    0, & \text{otherwise},
    \end{cases}
\end{align*}
\]

where a contribution of \(-\mu\) to \(A_\alpha\) here denotes a contribution of \(\mu\) to \(A_\beta\). To see this, note that a set of the 4 cuts corresponding to the output 1110 of \(C_i\) would lie respectively:

- To the right of the leftmost valuation block of Agent \(\alpha_1\) in \(R^\text{out}_i\), thus labelling the whole block \(A_+\).
- To the right of the rightmost valuation block of Agent \(\alpha_1\) in \(R^\text{out}_i\), thus labelling the whole block \(A_+\).
- To the right of the leftmost valuation block of Agent \(\alpha_2\) in \(R^\text{out}_i\), thus labelling the whole block \(A_+\).
- To the left of the leftmost valuation block of Agent \(\alpha_2\) in \(R^\text{out}_i\), thus labelling the whole block \(A_+\).

Since all of these valuation blocks have volume \(1/400\) each, the total contribution to \(A_+\) from an output of 1110 (and therefore a label of 1) is \(2/400\) for Agent \(\alpha_1\), whereas for Agent \(\alpha_2\), the total contribution is 0 and the sub-partition restricted to \(R^\text{out}_i\) is balanced. The argument for the remaining output/labels is very similar.

The "wrap-around" labels: In some cases, the circuit-encoders \(C_i\) will detect points close to the boundary of the triangular domain of \(\text{VARIANT TUCKER}\) in which case the sequence of points \(1, \ldots, 100\) extracted from the bit-extractors of the circuits will be part of a "wrap-around" line segment, i.e. a line segment that starts with some point with \(y\) close to zero and ends with a point with \(x\) close to 0 (i.e. it crosses the bottom boundary of the triangle region). In this case, Definition 3.5 requires that the "equal-and-opposite" property holds after we flip the labels of the wrapped-around subsequence.

In terms of \(I_{CH}\), this situation occurs when (i) either there are two cuts \(c_1\) and \(c_2\) in the c-e region and \(c_2\) sits very close to 1 or (ii) when there is only one cut in the c-e region (which can be thought of as another cut being situated exactly at \(0\)). In either case, since each circuit encoder \(C_i\) detects a virtual cut, (which is a shifted version of the cut detected by \(C_{i-1}\) as explained earlier), this sequence of points will be correctly generated by the reduction. For example, where there is only one cut \(c\) in the c-e region, while the bit-extractors of \(C_i\) only "see" that cut, the bit-extractors of each circuit-encoder \(C_2, \ldots, C_{\text{100}}\) "see" another cut, situated at position \(i \cdot 2^{-n-1}\). This is because the "wrapped-around" valuation of the first \(n + 8\) c-e identical sensors of \(C_i\) "sees" both \(A_+\) (on the left side of \(c\)) and \(A_-\) (on the right side of \(c\)), and therefore detect that a cut intersects the region - this is the virtual cut \(c_v\) detected by \(S_i\) (similarly for the case of two cuts).

Interpreting the virtual cut \(c_v\) as the actual cut, the circuit-encoder \(C_i\) now "sees" the label \(A_\beta\) on the left-hand side of \(c_v\) and \(A_\alpha\) on the right-hand side. Intuitively, \(C_i\) interprets the input as if the left endpoint of the region was \(1 - i \cdot 2^{-n-1}\) (i.e. as if we cut the c-e region at the point where the wrap-around value starts and glued the cut piece to the end of the c-e region), with the sequence of labels starting with \(A_\alpha\). The pre-processing circuit \(C_{\pre}\) ensures that the correct point of the wrapped-around subsequence is encoded, and the XOR operator circuit of \(C_i\) flips the label of this point, as desired by Definition 3.5.

5 PROOF OF THE REDUCTION

In this section, we prove the correctness of the reduction, i.e. that given a solution \(\mathcal{H}\) of \(\epsilon\)-Consensus-Halving, we can recover a solution to \(\text{VARIANT TUCKER}\) (and therefore to \(\text{TUCKER}\), given our results in Section 3). The main result of this section is the following.

**Theorem 5.1.** \(\text{VARIANT TUCKER}\) is polynomial-time reducible to Consensus-Halving.

Let \(C_i\), be one of the 100 copies of the circuit \(C\) in an instance \(I_{CH}\) of \(\epsilon\)-Consensus-Halving as constructed in Section 4. We say that \(C_i\) receives good inputs with respect to positions \((x, y)\) of the c-e cuts, if \(C_i\) receives valid boolean-encoding inputs extracted from \(x\) and \(y\). For example, in the case of \(i = 1\), \(C_1\) receives good inputs provided that the point \((x, y)\) of the domain of \(\text{VARIANT TUCKER}\) is not too close to the boundary of a sub-region.

The following observation is based on the density of the domain of \(\text{VARIANT TUCKER}\). Given the resolution used for the grid points within the square regions, there can be at most 4 points that are very close to the boundary of a sub-region.

**Observation 2.** At most 4 copies of \(C\) do not receive good inputs.

Using Lemma 4.1 (stated in Section 4.4.1), we can now prove the following lemma regarding the number of cuts in the c-e region, in any solution to \(I_{CH}\).

**Lemma 5.2.** In a solution to an instance \(I_{CH}\) of \(\epsilon\)-Consensus-Halving constructed as in Section 4, the two c-e cuts are the only cuts that may occur in the c-e region, and at least one c-e cut must occur in the c-e region.

**Proof.** To see this, note first that in any solution \(\mathcal{H}\) to \(I_{CH}\), all cuts apart from the c-e cuts, are constrained to lie in various intervals outside the c-e region. In particular, for every agent \(\alpha_i \in \mathcal{A}\) (i.e. every agent besides the two c-e agents \(\alpha_1, \alpha_2\)), it holds that most of the valuation of the agent (in particular, sufficiently more than a \((1/2)\)-fraction) lies in a designated interval, which we will denote by \(R_{\alpha_i}\). Agent \(\alpha_i\) is not the only agent that has non-zero value in \(R_{\alpha_i}\), but it holds that for \(j \neq i\), \(R_{\alpha_i} \cap R_{\alpha_j} = \emptyset\) i.e. each agent in \(\mathcal{A}\) has a different designated interval. Also, note that none of these intervals intersects with the c-e region, i.e. \(R_{\alpha_i} \cap \{0, 1\} = \emptyset\) for all agents \(\alpha_i \in \mathcal{A}\).

Obviously, by construction, for such an interval \(R_{\alpha_0}\), if there is no cut that intersects the interval, then agent \(\alpha_0\) will be dissatisfied with the balance of \(A_+\) and \(A_-\) and \(\mathcal{H}\) will not be a solution to \(I_{CH}\). Additionally, since there are \(N - 2\) such designated intervals which do not intersect with the c-e region, \(\mathcal{H}\) must place at most 2 cuts in the c-e region. This establishes the first statement of the Lemma.

Now for the second statement, suppose that neither c-e cut lies in the c-e region, in which case the c-e region gets labelled entirely \(A_+\). By Lemma 4.1, the blanket sensor agents will detect the discrepancy and \(\mathcal{H}\) can not be a solution.

**Proof of Proposition 4.3.** We will argue for the c-e identical bit extractors with value in \([0, 1/8]\); the argument for the rest is similar. First of all, note that in a solution to \(I_{CH}\) there can be at most
one cut intersecting the interval \([0, 1/8]\), otherwise the blanket-sensor agent would not be satisfied, by Lemma 4.1. Assume that such a cut \(c\) intersects the interval \([0, 1/8]\). To recover the position, Agent \(\alpha_{c}^{k}\) is responsible for determining whether the cut lies in the first or the second half of \([0, 1/8]\). If the cut lies in the first half, then the bit-detection gadget of the agent in \([\ell(R), \ell(R)+1/20]\) will detect a \(0\), with a cut intersecting (or sitting close to) the leftmost thin valuation block of its bit-detection gadget. This follows by the construction, since the cuts that intersect the outputs of the bit-extractors in \(R^{\ell}_{1}\) have \(A_{\alpha}\) on their left-hand side.

In turn, Agent \(\alpha_{c}^{k}\) will determine whether the cut lies in the first or the second half of the previously detected \(H\) and the bit will be set accordingly, with the corresponding cut lying on the left thin valuation block of the bit-detection gadget (case of 0) and on the right valuation block (case of 1). This is achieved with the extra small block of valuation \(1/20\) in \([\ell(R) + 0.25, \ell(R) + 0.75]\), which has already been labelled by the cut that intersects the output of Agent \(\alpha_{c}^{k}\). One can view this as adding a “compensation” to the position that is not in excess for the second agent (e.g. more \(A_{\alpha}\) assuming the first detected bit was 0), compared to the first agent. In particular, while the bit-detection gadget of the first agent uses a bit to detect the “direction” of the discrepancy, the bit-detection gadget of the second agent uses a bit to determine the direction of the discrepancy if additional value of 0.05 is added to the lesser label. The argument for the agents \(\alpha_{i}^{k}\) for \(k = 1, \ldots, n + 8\) is similar.

It should be noted that for the other copies \(C_{i}, i = 1, \ldots, 100\) of the circuit, the valuation blocks of the \(c\)-\(e\) identical agents in the \(c\)-\(e\) region might “wrap around”, i.e. they can consist of valuation blocks in \([0, z_{2}]\) and \([z_{2}, 1]\) where \([0, z_{2}] \cup [z_{1}, 1] = 1/8\). In that case, exactly the same arguments apply considering the interval to be \([z_{1}, z_{2}]\), i.e. the first half of the interval is \([z_{1}, z_{1} + 1/4]\) if \(z_{1} + 1/4 \leq 1\) and \([z_{1}, 1] \cup [0, z_{2} - 1/4]\) if \(z_{1} + 1/4 > 1\).

**Proof of Proposition 4.4.** Consider a set \(S_{i}^{j} \subseteq S_{j}\) of \(c\)-\(e\) identical agents with value in \([j/8, (j + 1)/8]\) of the \(c\)-\(e\) region, for some \(j \in \{0, \ldots, 7\}\). Assume first that a cut lies in \([j/8, (j + 1)/8]\) and that no other cut lies in \([0, j/8]\). Then, (since by convention the first cut in the \(c\)-\(e\) region has \(A_{\alpha}\) on its left-hand side), the \(n + 8\) \(c\)-\(e\) identical agents of region \([j/8, (j + 1)/8]\) will detect the position of the cut in the interval and their outputs will feed that to the gate-agents, exactly as described for the \(c\)-\(e\) identical agents of \([0, 1/8]\) in Section 4.4.1, and according to Proposition 4.3.

Now assume that that the second cut in the \(c\)-\(e\) region lies in \([j/8, (j + 1)/8]\) and the first cut lies somewhere in \([0, j/8]\). Observe that the first cut must have been detected by another set \(S_{i}^{j}\) of \(c\)-\(e\) identical bit-extractors, with \(j' < j\). Since the agents in \(S_{i}^{j}\) are now extracting the position of the second cut, notice that the label on the left-hand side of the cut is now \(A_{\alpha}\), which effectively “flips” the outputs of the bit-extractors (the bit-detection gadgets) \(S_{i}^{j}\) in \(R^{\ell}_{1}\). However, since all this information is provided to the pre-processing circuit, the circuit can infer how to interpret the outputs (and particularly it can lead the outputs of the set \(S_{i}^{j}\) through a set of NOT gates).

In simpler words, if a cut has already been detected by a set of sensors, this informs the circuit on how to interpret the remaining inputs that correspond to the second cut. Similarly, the circuit can use the information that no cuts occur in the region \([j/8, (j + 1)/8]\), which will be either a string of 1s (if no cut has been detected in a previous interval) or a string of 0s (if a cut has been detected in a previous interval). Since the circuit knows whether a cut has been detected in an interval \([j'/8, (j' + 1)/8]\), with \(j' < j\), it also knows how to interpret these trivial inputs. Finally, the pre-processing circuit \(C_{i}^{\mathrm{pre}}\) can combine the inputs from all the different intervals into a \((n + 4)\)-bit string which encodes the coordinates of a point \((x, y)\) in the domain.

**5.1 Dealing with the Stray Cut**

As we mentioned in Section 1, all agents other than the coordinate-encoding ones are associated with separate cuts. For all the circuit-encoding agents, these cuts are constrained to lie in different regions in \(R\), but for the \(c\)-\(e\) agents, it is not a priori clear that these cuts will lie in the \(c\)-\(e\) region. Lemma 5.2 establishes that in any solution of \(I_{CH}\), at least one of these cuts will actually lie in the \(c\)-\(e\) region, but the other might actually move into the circuit-encoding region \(R\). We will refer to such a cut as a *stray cut*. A stray cut may have two effects on \(H\), it can intersect the circuit-encoding region \(R_{c}\) of some circuit encoder \(C_{i}\), for \(i \in \{1, \ldots, 100\}\) and it can flip the parity of the circuit encoders \(C_{i}\) with \(R_{c} < c\), where \(c\) is the position of the stray cut in \(R_{c-1}\). In other words, if the first cut \(R_{1}\) was expecting to see \(A_{\alpha}\) on its left-hand side, it now sees \(A_{\alpha}\) and vice-versa.

The first effect is not much of a problem; we simply deem this circuit “unreliable”. Since there is only one stray cut, there is at most one unreliable circuit \(C_{i}\). The error that this copy will introduce to the volumes of the labels \(A_{\alpha}\) for the \(c\)-\(e\) agents (see Section 4.6) will be relatively small due to the fact that there are many reliable points that receive good inputs.

The second effect from the ones above is potentially more troublesome however, since the parity flip could alter the outputs of the bit-extractors. This problem however is actually being taken care of by the pre-processing circuit (and the XOR operator of the main circuit). If outputs of the bit-extractors are flipped, the pre-processing circuit actually inputs the bit-wise complements of the raw data that it would input before the flip. The effects of these flips cancel out and the circuit outputs exactly the same label, which is then flipped by the XOR sub-circuit, to ensure that the \(c\)-\(e\) agents receive the same feedback.

Due to lack of space, we leave the details for the full version.

**5.2 Correctness Lemmas**

By Lemma 5.2, we are left with two cases to consider: the first case when both \(c\)-\(e\) cuts lie in the \(c\)-\(e\) region, and the second case when just one of the lies in the \(c\)-\(e\) region.

**Lemma 5.3.** Consider a solution \(H\) to \(I_{CH}\) in which both \(c\)-\(e\) cuts lie in the \(c\)-\(e\) region and a set of points \((x_{1}, y_{1}), \ldots, (x_{100}, y_{100})\) recovered from \(H\) as described in Section 4.5. Then \((x_{1}, y_{1}), \ldots, (x_{100}, y_{100})\) is a solution to Variant Tucker.

**Proof.** Let \(c_{1}(z_{1})\) and \(c_{2}(z_{2})\) be the positions of the \(c\)-\(e\) cuts, which are assumed in the statement of the lemma to both lie in \([0, 1]\). Since there are two cuts in the \(c\)-\(e\) region, by the recovery of the solution to Variant Tucker, we have \(x = c_{1}(z_{1})\) and \(y = 1 - c_{2}(z_{2})\).
and the sequence of 100 points that is a solution to $I_{VT}$ consists of $(x_1, y_1), (x_1 + 2^{-n} + 1, y_1 - 2^{-n} + 1), \ldots (x_1 + 99\cdot 2^{-n} + 1, y_1 - 99\cdot 2^{-n} + 1)$ where addition/subtraction are taken modulo 1.

By construction of the solution according to Section 4.5 and by the resolution of the domain, the bit extractors of $C_1$ extract the binary representation of the coordinates $(x_1, y_1)$, according to Proposition 4.3 in Subsection 4.4.1. Then, as explained in Section 4.4.2 and Proposition 4.5, these coordinates are propagated via the gate agents in $G_k$ and correspond to an output of $C_1$ (a bit-string of length 4, where there is a one-to-one correspondence between the labels $\{-1, 1\}$ and 4 distinguished output bit-strings, namely 0001, 1110, 0110, 1011 respectively).

Since each copy of the circuit in the c-e region is a shifted version of the previous copy by $2^{-n+1}$, it is not hard to see that the bit extractors of a reliable circuit $C_1$ that receives good inputs, actually detect the representation of point $(x_1, y_1)$ in the sequence of 100 points originating with $(x_1, y_1)$. In precisely the same way, the output of this circuit feeds a discrepancy back to the c-e agents. Therefore, in a solution $\mathcal{H}$ to $IC_{CH}$, the points that are detected from the bit extractors of the circuits $C_1, \ldots, C_{100}$ will actually correspond to the points in the sequence $(x_1, x_2), \ldots, (x_{100}, y_{100})$.

As explained in Section 4.6, each such output string corresponds to a labelling of the valuations of the c-e agents in $R_{out}$ (the volumes of $A_+/A_-$ are balanced in $R_{in}$, since the blanket sensor agent $a_{1+}$ is passive) and therefore there is a discrepancy in $R_{out}$ for exactly one c-e agent. Specifically, for Agent $a_j$, with $j \in \{1, 2\}$, the discrepancy is in favour of $G_k$, with $k \in \{+, -, 0\}$, if the label of the output is $kj$.

One can easily check that for a c-e agent to be satisfied with the balance of the labels, it has to be the case that the excess in $A_+$ or $A_-$ due to a specific output in region $R_{out}^j$ has to be "cancelled out" from an excess of the opposite label ($A_+$ or $A_-$ respectively) in another interval $R_{out}^j \subseteq R_i$. For this to be possible, by construction, it has to be the case that the output of the corresponding circuit $C_j$ corresponds to the opposite label of the output of $C_i$, if that copy of the circuit operates as intended. Therefore, if the points $(x_1, y_1)$ and $(x_j, y_j)$ are detected by reliable copies $C_i$ and $C_j$ that receive good inputs, they must have coordinates in different tiles of the domain, which are labelled with opposite labels. However, by the density of the domain and since there are at least 100 points of the domain in the line-segment between $(x_1, y_1)$ and $(x_j, y_j)$, this is only possible if these points lie in neighbouring tiles of equal and opposite labels, i.e. in a solution to VARIANT TUCKER.

A degenerate case occurs when some of the points $(x_i, y_i)$ in the sequence correspond to circuits that do not achieve good inputs (note that since both $c(a_1)$ and $c(a_2)$ lie in the c-e region, there are no stray cuts by definition). These are the points that lie close to the boundary of two tiles and their labels assigned by the circuit are unconstrained. This in principle can cause a cancellation effect and "balance out" the discrepancies of some unambiguously labelled point, when both of these points lie in the same tile (the former near the boundary and the latter in the interior). For example, for a point $p_1$ labelled $-1$ in some tile $j$, there can be a point $p_2$ close to the boundary with some neighbouring tile $j'$ (with tile $j'$ labelled $-1$ as well), that receives label 1 by the circuit (due to the fact that the labelling rules of boundary points are unconstrained). In a sequence of points that contain both $p_1$ and $p_2$, the $A_+/A_-$ discrepancy due to $p_1$ will cancel out the $A_+/A_-$ discrepancy due to $p_1$, although we are not at a solution.

This is being take care of by the averaging manoeuvre, which uses 100 copies of the circuit and requires that at least 10 of the points in the sequence receive a label and 10 other points receive an equal and opposite label. More concretely, assume by contradiction that we are at a solution $\mathcal{H}$ of $IC_{CH}$, but the sequence of 100 points do not correspond to a solution to $I_{VT}$. Let $\lambda$ be the label of the majority of the points in the sequence (breaking ties arbitrarily) and assume wlog that $\lambda = 1$. Observe that by the chosen resolution of the domain, it holds that at least 40 points in the sequence must be labelled 1. By the discussion above, since $\mathcal{H}$ is a solution, for every point labelled 1, there must be another point in the sequence labelled $-1$, for the cancellation to take place. By Observation 2, there are at most 4 such points that are arbitrarily labelled and therefore they can contribute to a cancellation of at most 1/10 of the excess of $A_+$ due to the contribution of the points labelled 1. This means that there must be at least 36 points labelled $-1$ in the sequence and the sequence $(x_1, y_2), \ldots, (x_{100}, y_{100})$ is actually a solution to $I_{VT}$.

\textbf{Lemma 5.4.} Consider a solution $\mathcal{H}$ to $IC_{CH}$ in which only one c-e cut lies in the c-e region and a set of points $(x_1, y_1), \ldots, (x_{100}, y_{100})$ recovered from $\mathcal{H}$ as described in Section 4.5. Then $(x_1, y_1), \ldots, (x_{100}, y_{100})$ is a solution to VARIANT TUCKER. \hfill $\square$

\textbf{Proof.} The proof is very similar to the proof of Lemma 5.3. Here, if $c$ is the position of the single cut in the c-e region, we have that $x = 0$ and $y = 1 - c$. Again, the binary expansion of $(x, y)$ is extracted from the bit extractors of $C_1$ and the output of the encoded circuit will correspond to a discrepancy for the c-e agents in $R_{out}$ similarly as before. The same is true for the remaining 99 circuits, with the exception of possibly one circuit that might be unreliable due to the stray cut. From the discussion in Section 5.1, it holds that the feedback of any reliable copy to the c-e agents is unaffected by the stray cut.

A stray cut intersecting interval $R_i$ might introduce some additional discrepancy in the volume of the two labels in $R_i$, which is upper bounded by the valuation of the coordinate-encoding agents in $R_i$, i.e. $1/100$. The effect that this could have is that this discrepancy might cancel out the discrepancies in favour of $A_+$ or $A_-$ introduced by at most 3 reliable circuits that receive good inputs (which happens if all of the valuation of the c-e agent in $R_i$ is labelled $A_+$ or $A_-$ respectively).

However, similarly to before, this can "invalidate" at most 7 points overall and there will still be 33 points labelled 1 whose contribution to $A_+$ needs to be cancelled out by points labelled $-1$ and we will be at a solution to $I_{VT}$.

\section{6 Equivalence of Consensus Halving and Necklace Splitting}

In this section, we show that approximate Consensus Halving and the well-known Necklace Splitting problem \cite{halving} are computationally equivalent, i.e. the reduce to each other in polynomial time.

\textbf{Definition 6.1 (Necklace Splitting \cite{halving}).} In the necklace splitting problem, there is an open necklace (an interval) with $k \cdot m$ beads, each of which has one of $n$ colours. There are $a_i \cdot k$ beads of colour
We hope that the present work will lead to more PPA-completeness from inverse-exponential to inverse-polynomial, we should lose version: we start from a solution to consensus halving and prove
Necklace-splitting the case if PPA-hardness but retain PPAD-hardness. That is what would be
Necklace-splitting for believing that results, starting with the Necklace Splitting problem. The reason
7 CONCLUSIONS
from Consensus Halving to Necklace splitting, the heart of the
We leave the details of the proof of the full version. For reducing from Consensus Halving to Necklace Splitting, the heart of the
argument is “simulating” the valuation blocks by beads, where each bead corresponds to a certain volume of value and then arguing how a set of cuts for Necklace Splitting can solve the former problem. For the other direction, we use a very similar idea to the one presented by Alon [2] for proving that a solution to (discrete) Necklace Splitting can be obtained from a solution for the continuous version: we start from a solution to consensus halving and prove using induction that it can be transformed to a necklace splitting solution, by appropriately moving some of the cuts, if necessary.

Theorem 6.2. e-consensus-halving and Necklace Splitting are computationally equivalent, when ϵ is inverse-polynomial. This implies that Necklace Splitting is PPAD-hard.

We leave the details of the proof of this theorem.

7 CONCLUSIONS
We hope that the present work will lead to more PPA-completeness results, starting with the Necklace Splitting problem. The reason for believing that Necklace-splitting is PPA-complete, is that it would be surprising if, in relaxing the approximation parameter ϵ from inverse-exponential to inverse-polynomial, we should lose PPA-hardness but retain PPAD-hardness. That is what would be the case if Necklace-splitting were merely PPA-complete.

REFERENCES