


Strong Approximate Consensus Halving and the Borsuk-Ulam Theorem

Eleni Batziou ✉

Technical University of Munich, Germany

Kristoffer Arnsfelt Hansen ✉ 

Aarhus University, Denmark

Kasper Høgh ✉

Aarhus University, Denmark

Abstract

In the consensus halving problem we are given n agents with valuations over the interval $[0, 1]$. The goal is to divide the interval into at most $n + 1$ pieces (by placing at most n cuts), which may be combined to give a partition of $[0, 1]$ into two sets valued equally by all agents. The existence of a solution may be established by the Borsuk-Ulam theorem. We consider the task of computing an approximation of an exact solution of the consensus halving problem, where the valuations are given by distribution functions computed by algebraic circuits. Here approximation refers to computing a point that is ε -close to an exact solution, also called *strong* approximation. We show that this task is polynomial time equivalent to computing an approximation to an exact solution of the Borsuk-Ulam search problem defined by a continuous function that is computed by an algebraic circuit.

The Borsuk-Ulam search problem is the defining problem of the complexity class BU. We introduce a new complexity class BBU to also capture an alternative formulation of the Borsuk-Ulam theorem from a computational point of view. We investigate their relationship and prove several structural results for these classes as well as for the complexity class FIXP.

2012 ACM Subject Classification Theory of computation → Complexity classes; Theory of computation → Problems, reductions and completeness

Keywords and phrases Consensus halving, Computational Complexity, Borsuk-Ulam

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.24

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: <https://arxiv.org/abs/2103.04452>

Funding *Kristoffer Arnsfelt Hansen and Kasper Høgh*: Supported by the Independent Research Fund Denmark under grant no. 9040-00433B.

Eleni Batziou: Supported by the German Research Foundation (DFG) within the Research Training Group AdONE (GRK 2201).

1 Introduction

Many computational problems, e.g. linear and semidefinite programming, are most naturally expressed using real numbers. When the model of computation is discrete, these problems must be recast as discrete problems. In the case of linear programming this causes no problems. Namely, when the input is given as rational numbers and an optimal solution exists, a rational valued optimal solution exists and may be computed in polynomial time. For semidefinite programming however, it may be the case that all optimal solutions are irrational. For dealing with such cases we may instead consider the *weak optimization* problem as defined by Grötschel, Lovász and Schrijver [18]: Given $\varepsilon > 0$, the task is to compute a rational-valued vector x that is ε -close to the set of feasible solutions and has objective value ε -close to optimal. Assuming we are also given, as an additional input, a strictly feasible



© Eleni Batziou, Kristoffer Arnsfelt Hansen, and Kasper Høgh;
licensed under Creative Commons License CC-BY 4.0

48th International Colloquium on Automata, Languages, and Programming (ICALP 2021).

Editors: Nikhil Bansal, Emanuela Merelli, and James Worrell; Article No. 24; pp. 24:1–24:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



solution and a bound on the magnitude of the coordinates of an optimal solution, the weak optimization problem may be solved in polynomial time using the ellipsoid algorithm [18]. Let us note however that without additional assumptions, even the complexity of the basic *existence problem* of semidefinite feasibility is unknown. In fact, the problem is likely to be computationally very hard [23]. More precisely, it is hard for the problem PosSLP, which is the fundamental problem of deciding whether an integer given by a division free arithmetic circuit is positive [2].

In this paper we consider real valued search problems, where existence of a solution is guaranteed by topological existence theorems such as the Brouwer fixed point theorem and the Borsuk-Ulam theorem. This means that the search problems are *total*, thereby fundamentally differentiating them from general search problems where, as described above, even the existence problem may be computationally hard. We are mainly interested in the *approximation* problem: given $\varepsilon > 0$, the task is to compute a rational-valued vector x that is ε -close to the set of solutions.

Recall that the Brouwer fixed point theorem states that every continuous function $f: B^n \rightarrow B^n$, where B^n is the unit n -ball, has a fixed point, i.e. there is $x \in B^n$ such that $f(x) = x$ [6]. The Borsuk-Ulam theorem states that every continuous function $f: S^n \rightarrow \mathbb{R}^n$, where S^n is the unit n -sphere in \mathbb{R}^{n+1} , maps a pair of antipodal points of S^n to the same point in \mathbb{R}^n , i.e. there is $x \in S^n$ such that $f(x) = f(-x)$ [5]. The Brouwer fixed point theorem is of course not restricted to apply to the domain B^n , but applies to any domain that is homeomorphic to B^n . Similarly the Borsuk-Ulam theorem applies to any domain homeomorphic to S^n by an antipode-preserving homeomorphism. It is well-known that the Borsuk-Ulam theorem generalizes the Brouwer fixed point theorem, in the sense that the Brouwer fixed point theorem is easy to prove using the Borsuk-Ulam theorem [22, 24].

The Brouwer fixed point theorem and the Borsuk-Ulam theorem naturally define corresponding real valued search problems, and thereby also corresponding approximation problems. In addition, the statements of the theorems naturally lead to another notion of approximation. For the case of the Brouwer fixed point theorem we may look for an *almost* fixed point, i.e. $x \in B^n$ such that $f(x)$ is ε -close to x , and for the case of the Borsuk-Ulam theorem we look for a pair of antipodal points that *almost* map to the same point, i.e. $x \in S^n$ such that $f(x)$ and $f(-x)$ are ε -close. Following [12], we shall refer to this notion of approximation as *weak* approximation and to make the distinction clear we refer to the former (and general) notion of approximation as *strong* approximation. In the setting of weak approximation in relation to the Borsuk-Ulam theorem we assume that f has domain B^n .

In their seminal work, Etessami and Yannakakis [12] introduced the complexity class FIXP to capture the computational complexity of the real-valued search problems associated with the Brouwer fixed point theorem, and proved that the problem of finding a Nash equilibrium in a given 3-player game in strategic form is FIXP-complete. In order to have a notion of completeness, the class FIXP is defined to be closed under reductions. The type of reductions chosen by Etessami and Yannakakis, SL-reductions, consists of mapping between sets of solutions by a composition of a *projection* reduction followed by individual affine transformations applied to each coordinate.

Etessami and Yannakakis considered different ways to cast real valued search problems as discrete search problems. In addition to the approximation problem, these are the *partial computation* problem where the task is to compute a solution to a given number of bits of precision and *decision* problems, where the task is to evaluate a sign condition of the set of solutions given the promise that either all solutions satisfy the condition or none of them do. Of these we shall only consider the approximation problem. The class FIXP_a denotes the class of discrete search problems corresponding to strong approximation of Brouwer

fixed points and is defined to be closed under polynomial time reductions. Etessami and Yannakakis also prove that the problem PosSLP reduces to the problem of approximating a Nash equilibrium, thereby showing that FIXP_a likely contains search problems that are computationally very hard.

While the notion of SL-reductions is very restricted, it is sufficient for proving completeness of the problem of finding a Nash equilibrium. Likewise, SL-reductions are sufficient for showing that FIXP is robust with respect to the choice of domain for the Brouwer function.

Another important reason for using SL-reductions is that they immediately imply polynomial time reductions between the corresponding decision and approximation problems (the partial computation problem is more fragile and requires additional assumptions, cf. [12]). As we are mainly interested in the approximation problem more expressive notions of reducibility can be considered, while maintaining the property that reducibility implies polynomial time reducibility between the corresponding approximation problems. A sufficient condition for this is that the mapping of solutions is *polynomially continuous* and polynomial time computable.

1.1 The Borsuk-Ulam Theorem

Deligkas, Fearnley, Melissourgos, and Spirakis [11] recently introduced the complexity class BU to capture, in an analogy to FIXP , the computational complexity of the real-valued search problems associated with the Borsuk-Ulam theorem.

The Borsuk-Ulam theorem has a number of equivalent statements that are also easy to derive from each other. A function f defined on the unit sphere S^n is *odd* if $f(x) = -f(-x)$ for all $x \in S^n$. Note that the boundary ∂B^n of the unit n -ball B^n is identical to S^{n-1} . We thus say that a function f defined on B^n is odd on ∂B^n if f is odd when restricted to S^{n-1} . We present the simple proof of the known fact that the different formulations can be derived from each other, for the purpose of discussing equivalence from a computational point of view.

► **Theorem 1** (Borsuk-Ulam). *The following statements hold:*

- (1) *If $f: S^n \rightarrow \mathbb{R}^n$ is continuous there exists $x \in S^n$ such that $f(x) = f(-x)$.*
- (2) *If $g: S^n \rightarrow \mathbb{R}^n$ is continuous and odd there exists $x \in S^n$ such that $g(x) = 0$.*
- (3) *If $h: B^n \rightarrow \mathbb{R}^n$ is continuous and odd on ∂B^n there exists $x \in B^n$ such that $h(x) = 0$.*

Proof of equivalence. Given f we may define $g(x) = f(x) - f(-x)$. Clearly g is odd and we have $g(x) = 0$ if and only if $f(x) = f(-x)$, which shows that (2) implies (1). Conversely, given g we simply let $f = g$. If $f(x) = f(-x)$, then since g is odd we have $f(x) = g(x) = -g(-x) = -f(-x) = -f(x)$ and hence $g(x) = f(x) = 0$, which therefore shows that (1) implies (2).

We may view S^n as two hemispheres, each homeomorphic to B^n , which are glued together along their equators. Let $\pi: S^n \rightarrow B^n$ be the orthogonal projection defined by $\pi(x_1, \dots, x_{n+1}) = (x_1, \dots, x_n)$. Then given h we may define $g(x) = h(\pi(x))$ for $x_{n+1} \geq 0$ and $g(x) = -h(-\pi(x))$ for $x_{n+1} \leq 0$. The assumption that h is odd on ∂B^n makes g a well-defined continuous odd function. We have $g(x) = 0$ if and only if $h(x) = 0$, which shows that (2) implies (3). Conversely, given g we define h by $h(x) = g\left(x, (1 - \|x\|_2^2)^{\frac{1}{2}}\right)$. Then h is continuous and odd on ∂B^n , since $x \in \partial B^n$ if and only if $\|x\|_2^2 = 1$. Clearly if $h(x) = 0$ we may let $y = (x, (1 - \|x\|_2^2)^{\frac{1}{2}})$ and have $g(y) = 0$. On the other hand, when $g(y) = 0$ we may define $x = (y_1, \dots, y_n)$ if $y_{n+1} \geq 0$ and $x = (-y_1, \dots, -y_n)$ if $y_{n+1} < 0$, and we have $h(x) = 0$. Together this shows that (3) implies (2). ◀

The class BU defined in [11] corresponds to the first formulation of the above theorem. We may clearly consider the second formulation equivalent to the first also from a computational point of view. In particular, when translating between formulations, the set of solutions is unchanged. Note that this set of solutions has the property that all solutions come in pairs: when x is a solution then $-x$ is a solution as well. For the third formulation of the theorem this property only holds for solutions on the boundary ∂B^n .

In contrast, while the mapping of solutions of the third formulation to the second (and first) formulation given above is continuous this is not the case in the other direction. More precisely, consider $y \in S^n$ such that $g(y) = 0$. For a solution strictly contained in the upper hemisphere, the orthogonal projection to the first n coordinates produces $x \in B^n$ such that $h(x) = 0$. For a solution y strictly contained in the lower hemisphere, the projection is instead applied to the antipodal solution $-y$.

To clarify this issue from a computational point of view we introduce a new class BBU of real valued search problems corresponding to the third formulation of Theorem 1, and it will follow from definitions that $\text{BU} \subseteq \text{BBU}$. In the context of strong approximation however, the corresponding classes of discrete search problems BU_a and BBU_a will be shown to coincide. The idea is that given an approximation to $y \in S^n$, where $g(y) = 0$, that is sufficiently close to the equator of S^n , there is no harm in *incorrectly* deciding to which hemisphere y belongs, since solutions $x \in \partial B^n$ for which $h(x) = 0$ also come in pairs.

For the class BU, the notion of SL-reductions is clearly too restrictive to allow a reasonable comparison to FIXP. Closing the class BU by SL-reductions, the solutions would still come in pairs, thereby imposing strong conditions on the set of solutions. On the other hand the reductions should also not be *too* strong. In particular it would be desirable that FIXP would still be closed under the chosen notion of reductions. This issue is not discussed in [11]. We shall therefore propose a suitable notion of reductions for both BU and BBU.

1.2 Consensus Halving

The Consensus halving problem is a classical problem of *fair division* [21]. We are given a set of n bounded and continuous measures μ_1, \dots, μ_n defined on the interval $A = [0, 1]$. The goal is to partition the interval A into at most $n + 1$ intervals, i.e. by placing at most n cuts, such that unions of these intervals form another partition $A = A^+ \cup A^-$ of A satisfying $\mu_i(A^+) = \mu_i(A^-)$ for every i . We may think of the intervals being assigned a *label* from the set $\{+, -\}$, and A^+ is precisely the union of the intervals labeled by $+$. Such a partition is also known as a consensus halving. Using the Borsuk-Ulam theorem, Simmons and Su [21] proved that a consensus halving using at most n cuts always exists. Simmons and Su represent a division of A as a point x on the unit n -sphere S_1^n with respect to the ℓ_1 -norm. The point x is viewed as representing a division into *precisely* $n + 1$ intervals, where some intervals are possibly empty. More precisely, the i -th interval has length $|x_i|$, and intervals of length 0 may simply be discarded. The intervals of positive length are then labeled according to $\text{sgn}(x_i)$. Note that for any x , the antipode $-x$ represents the division where the sets A^+ and A^- are exchanged. This naturally leads to a formulation using the Borsuk-Ulam theorem [21]. Namely we may consider the function $F: S_1^n \rightarrow \mathbb{R}^n$ given by $F(x)_i = \mu_i(A^+)$, and note that any $x \in S_1^n$ for which $F(x) = F(-x)$ represent a consensus halving.

We are interesting in the simple setting of additive measures, where we have corresponding density functions f_1, \dots, f_n such that $\mu_i(B) = \int_B f_i(x) dx$. To cast the consensus halving problem as a real valued search problem we follow [11] and assume that the measures μ_1, \dots, μ_n are given by the distribution functions F_1, \dots, F_n defined by $F_i(x) = \int_0^x f_i(t) dt$. An instance of the consensus halving problem is then given as a list of algebraic circuits computing these distribution functions.

1.3 Strong versus Weak Approximation

The difference between weak and strong approximation was studied in detail in the general context of the Brouwer fixed point theorem by Etessami and Yannakakis. A central example is the problem of finding a Nash equilibrium (NE). An important notion of approximation of a NE is the notion of an ε -NE. Computing an ε -NE of a given strategic form game Γ is polynomial time equivalent to computing a weak ε' -approximation to a fixed point the Nash's Brouwer function F_Γ associated to Γ [12, Proposition 2.3]. In turn, computing a weak ε' -approximation to a fixed point of F_Γ polynomial time reduces to computing a strong ε'' -approximation to a fixed point of F_Γ [12, Proposition 2.2], since the function F_Γ is polynomially continuous and polynomial time computable. In general however an ε -NE might be far from any actual NE, unless ε is inverse doubly exponentially small as a function of the size of the game [12, Corollary 3.8].

For the problem of consensus halving we can illustrate the difference between weak and strong approximation by a simple example. We shall refer to a weak ε -approximation of a consensus halving as simply an ε -consensus halving. Consider a single agent whose measure μ on the interval $[0, 1]$ is given by the following density

$$f(x) = \begin{cases} (1 + \varepsilon)/\varepsilon & \text{if } 0 \leq x < \varepsilon/2 \\ 0 & \text{if } \varepsilon/2 \leq x < 1 - \varepsilon/2 \\ (1 - \varepsilon)/\varepsilon & \text{if } 1 - \varepsilon/2 \leq x \leq 1 \end{cases}$$

We have $\mu([0, 1]) = 1$ and since μ is a step function, the corresponding distribution function F is piecewise linear. The unique consensus halving is obtained by placing a cut at the point $\varepsilon/2 - \varepsilon^2/(2 + 2\varepsilon)$. Placing a cut at any point $t \in [\varepsilon/2 - \varepsilon^2/(1 + \varepsilon), 1 - \varepsilon/2]$ results in an ε -consensus halving, i.e. such that $|\mu([0, t]) - \mu([t, 1])| \leq \varepsilon$. Thus an ε -consensus halving might be very far from an actual consensus halving. Note also that placing a cut at any point $t \in [0, 3\varepsilon/2 - \varepsilon^2/(2 + 2\varepsilon)]$ is a strong ε -approximation, which illustrates that a strong approximation is not necessarily a weak approximation. On the other hand, a strong $(\varepsilon^2/2)$ -approximation is also an ε -consensus halving.

The Brouwer fixed point theorem and the Borsuk-Ulam theorem can both be proved starting from combinatorial analogues of the two theorems, namely from Sperner's lemma and Tucker's lemma, respectively. The proofs of these two lemmas are constructive, but using them to derive the Brouwer fixed point theorem and the Borsuk-Ulam theorem involve a nonconstructive limit argument. Let us note in passing that while Sperner's lemma, like the Borsuk-Ulam theorem, has several different formulations, it is usually formulated as the combinatorial analogue of the third formulation of Theorem 1.

Sperner's and Tucker's lemma give rise to total NP search problems. These turn out to be complete for the complexity classes PPAD and PPA introduced in the seminal work by Papadimitriou [19]. Papadimitriou proved PPAD-completeness of the problem given by Sperner's lemma as well as membership in PPA of the problem given by Tucker's lemma, while PPA-completeness of the latter problem was proved recently by Aisenberg, Bonet, and Buss [1]. These results also imply that the classes PPAD and PPA correspond to the problems of computing weak approximations to Brouwer fixed points and to Borsuk-Ulam points.

The computational complexity of the problems of computing an ε -NE and of computing an ε -consensus halving was settled in breakthroughs of two lines of research. Computing an ε -NE was shown to be PPAD-complete by Daskalakis, Goldberg and Papadimitriou [9] and Cheng and Deng [8]. Computing an ε -consensus halving was shown to be PPA-complete by Filos-Ratsikas and Goldberg [14, 15].

1.4 Our Results

Our main result is that the problem of strong approximation of consensus halving is equivalent to strong approximation of the Borsuk-Ulam theorem.

► **Theorem 2.** *The strong approximation problem for CH is BU_a -complete.*

As described, we view the consensus halving problem as the real valued search problem with its domain being either the unit sphere or the unit ball with respect to the ℓ_1 -norm. The theorem is proved by reduction from the real valued search problem associated with the Borsuk-Ulam theorem on the domain being the unit ball with respect to the ℓ_∞ -norm, i.e. from a defining problem of the class BBU.

It is of general interest to study the relationship between search problems given by the Borsuk-Ulam theorem on different domains from a computational point of view. The reduction establishing the proof of Theorem 2 gives additional motivation for this. The domains we consider are unit spheres S_p^n and unit balls B_p^n with respect to the ℓ_p -norm for $p \geq 1$ or $p = \infty$. It is of course straightforward to construct homeomorphisms between unit spheres or unit balls with respect to different norms, and these could be used to define reductions between the different problems. We would however like that the mapping of solutions is simple, and in particular we would like to avoid divisions and root operations. We prove that one may in fact reduce between domains using SL-reductions.

Deligkas et al. gave a reduction from the FIXP-complete problem of finding a Nash equilibrium to CH. Combined with membership of CH in BU, this gives the inclusion $\text{FIXP} \subseteq \text{BU}$. We observe that a proof due to Volovikov [24] of the Brouwer fixed point theorem from the Borsuk-Ulam theorem may be adapted to give a simple proof of the inclusion $\text{FIXP} \subseteq \text{BU}$.

For the class FIXP we prove two interesting structural properties that do not appear to have been observed earlier. While FIXP is defined using SL-reductions, we show that FIXP is closed under polynomial time reductions where the mapping of solutions is expressed by *general* algebraic circuits. This in particular supports that one may reasonably define the classes BU and BBU using less restrictive notions of reductions than SL-reductions. We propose to have the mapping of solutions be computed by algebraic circuits involving the operations of addition, multiplication by scalars, as well as maximization. This means that the mapping of solutions is a piecewise linear function, and we refer to these as PL-reductions. The second structural result for FIXP is a characterization of the class by very simple Brouwer functions. These are defined on the unit-hypercube domain $[0, 1]^n$ and each coordinate function is simply one of the operations $\{+, -, *, \max, \min\}$, modified to have the output truncated to the interval $[0, 1]$.

For the classes BU and BBU we prove that they are also closed under reductions where the mapping of solutions is computed by general algebraic circuits, but with the additional requirement that this function must be odd.

For the class FIXP, an interesting consequence of the proof that finding a Nash equilibrium is complete, is that the class may be characterized by Brouwer functions computed by algebraic circuits without the division operation. The proof also shows that the class FIXP is unchanged even when allowing root operations as basic operations. We prove by a simple transformation that the classes BU and BBU may be characterized using algebraic circuits without the division operation. Furthermore, as a consequence of Theorem 2 the class of strong approximation problems $\text{BU}_a = \text{BBU}_a$ is unchanged even when allowing root operations as basic operations.

1.5 Comparison to previous work

As a precursor to the proof of PPA-completeness of computing an ε -consensus halving, Filos-Ratsikas, Frederiksen, Goldberg and Zhang [13] proved the problem to be PPAD-hard. Deligkas et al. [11] uses ideas from this proof together with additional new ideas to obtain their proof of FIXP-hardness for computing an exact consensus halving.

While $\text{PPAD} \subseteq \text{PPA}$, the PPAD-hardness result of [13] is not implied by the recent proofs of PPA-completeness. In particular, the work [13] proves PPAD-hardness even for *constant* ε , while the work of [15] only proves PPA-hardness for ε being inverse polynomially small. In the same way, while $\text{FIXP} \subseteq \text{BU}$, FIXP-hardness of computing an exact consensus halving is not implied by our reduction, since Theorem 2 establishes BU_a -hardness rather than BU-hardness. Recently a considerably simpler proof of PPA-hardness for computing an ε -consensus halving was given by Filos-Ratsikas, Hollender, Sotiraki and Zampetakis [16], and our reduction is inspired by this work.

All reductions described above are similar in the sense that one or more evaluations of a circuit are expressed in the consensus halving instance. The full interval A is partitioned into subintervals, cuts within these subintervals encode values in various ways, and agents implement the gates of the circuit by placing cuts. A main difference between the reductions establishing PPAD-hardness and FIXP-hardness to those establishing PPA-hardness is that in the former reductions, all cuts are constrained to be placed in distinct subintervals. The reason this is possible is that the objective is to find a fixed point of the circuit, which means that inputs and outputs may be identified.

In the setting of PPA and BBU the objective is to find a “zero” of the circuit. More precisely, for the setting of PPA the objective is to find two adjacent points of a given Tucker labeling that receive complementary labels, i.e. labels of different sign but same absolute value. For the setting of BBU the objective is to find an actual zero point of the circuit. All of the reductions establishing PPA-hardness of computing an ε -consensus halving have the property that cuts encoding the input of the circuit are *free* cuts, meaning that they can in principle be placed anywhere, and as a result also interfere with the evaluations of the circuit. This is also the case for our reduction, and this invariably limits its applicability to the approximation problem.

In the reduction of [16], the interval A is structured into different regions, a coordinate-encoding region, a constant-creation region, several circuit-simulation regions, and finally a feedback region. Our reduction also has a coordinate-encoding region and several circuit simulation regions, but the functions performed by the constant-creation region and feedback regions in [16] are in our reduction integrated into the individual circuit simulation regions and done differently.

A novelty of the reduction of [16] compared to previous reductions is in how values are encoded by cuts in subintervals. In previous reductions, values are encoded by what we will call *position encoding*. In that encoding it is required that there is exactly one cut in the subinterval, and the value encoded is determined by the distance between the cut position and the left endpoint of the interval. In [16] values are encoded by what we will call *label encoding*. Here there is no requirement on the number of cuts in the subinterval, and the value encoded is simply the difference between the Lebesgue measures of the subsets of the interval receiving label $+$ and label $-$. We shall employ a hybrid approach where the coordinate-encoding region uses label encoding while the circuit-simulation regions uses position encoding. The first step performed in a circuit-simulation region is thus to copy the input from the coordinate-encoding region. Switching to position encoding allows us in particular to implement a multiplication gate, similarly to [11]. Here the multiplication xy is

computed via the identity $xy = ((x + y)^2 - x^2 - y^2)/2$. In [11] where values range over $[0, 1]$, the squaring operation may be implemented directly by agents. In our case values range over the interval $[-1, 1]$, and the squaring operation is decomposed further, having agents compute it separately over the intervals $[-1, 0]$ and $[0, 1]$.

In analogy to [16] we have feedback agents that ensures that the circuit evaluates to 0 on the encoded input. The criteria that the agents check is however different, and for our purposes it is crucial that we have the same sign pattern in the position encoding of the output of the circuit as the copy of the input made by the circuit-simulation region. The actual detection of an output of 0 is performed by using approximations of the Dirac delta function. For computing the distribution functions of the feedback agents, we make use of the fact that these are computed by algebraic circuits, which enable us to make a strong approximation of the Dirac delta function via repeated squaring.

1.6 Organization of Paper

We introduce required terminology in Section 2. We refer to the full version of this paper for our structural results for the classes FIXP, BU, and BBU as well as the simple proof of the inclusion $\text{FIXP} \subseteq \text{BU}$. The proof of our main result, Theorem 2, is given in Section 3.

2 Preliminaries

2.1 Algebraic Circuits

Central to our work are algebraic circuits computing real valued functions. Let B be a finite set of real valued functions, for example $B = \{+, -, *, \div, \max, \min\}$. An *algebraic circuit* C with n inputs and m outputs over the *basis* B is given by an acyclic graph $G = (V, A)$ as follows. The *size* of C is equal to the number of nodes of G , which are also referred to as *gates*. The *depth* of C is equal to the length of the longest path of G . Every node of indegree 0 is either an *input gate* labeled by a variable from the set $\{x_1, \dots, x_n\}$ or a *constant gate* labeled by a real valued constant. Every other node is labeled by an element of B called the *gate function*. If a node v is labeled by a gate function $g: A \rightarrow \mathbb{R}$ with $A \subset \mathbb{R}^k$ we require that g has exactly k ingoing arcs with a linear order specifying the order of arguments to g . The output of C is specified by an ordered list of m (not necessarily distinct) nodes of G . The computation of C on a given input $x \in \mathbb{R}^n$ is defined in the natural way. Computation may fail in case a gate of C labeled by a function $g: A \rightarrow \mathbb{R}$ receives an input outside A , and in this case the output of C is undefined. Otherwise we say that the output is well defined and denote its value by $C(x)$. If $D \subseteq \mathbb{R}^n$ we say that C computes a function $f: D \rightarrow \mathbb{R}^m$ if $C(x)$ is well defined for all $x \in D$.

We shall in this paper just consider algebraic circuits where the basis consists only of continuous functions. This means in particular that any algebraic circuits computes a continuous function as well. We shall also only consider constant gates labeled with rational numbers. In this case we are also interested in the *bitsize* of the encoding of the constants, which is the maximum bitsize of the numerator or denominator.

By using multiplication with the constant -1 , the functions $-$ and \min may be simulated using $+$ and \max , respectively. In this way we may convert a circuit over the full basis $\{+, -, *, \div, \max, \min\}$ into an equivalent $\{+, *, \div, \max\}$ -circuit. We shall also consider circuits where use of the multiplication operator $*$ is *restricted* to having one of the arguments being a constant gate. We denote this by the symbol $*\zeta$ and use it in particular for defining $\{+, *\zeta, \max\}$ -circuits.

2.2 Search problems

A general search problem Π is defined by specifying to each input instance I a *search space* (or *domain*) D_I and a set $\text{Sol}(I) \subseteq D_I$ of *solutions*. We distinguish between *discrete* and *real-valued* search problems. For discrete search problems we assume that $D_I \subseteq \{0, 1\}^{d_I}$ for an integer d_I depending on I . Analogously, for real-valued search problems we assume that $D_I \subseteq \mathbb{R}^{d_I}$ for an integer d_I depending on I . One could likewise distinguish between search problems with discrete input and real-valued input. We are however mostly interested in problems where the input is discrete. That is, we assume that instances I are encoded as strings over a given finite alphabet Σ (e.g. $\Sigma = \{0, 1\}$).

Many natural search problems are however defined with a continuous search space. Not all of these may adequately be recast as discrete search problems, but are more naturally viewed as real-valued search problems. One approach for studying such problems would be to switch to the Blum-Shub-Smale model of computation [4]. A BSS machine resembles a Turing machine, but operates with real numbers instead of symbols from a finite alphabet. In particular is the input real-valued, and input instances are therefore encoded as real-valued vectors. All basic arithmetic operations and comparisons are unit-cost operations. One may then define real-valued analogues of Turing machine based classes. In particular, Blum, Shub and Smale defined and studied the real-valued analogues $P_{\mathbb{R}}$ and $NP_{\mathbb{R}}$ of P and NP . A BSS machine may in general make use of real-valued *machine constants*. If a BSS machine only uses rational valued machine constants we shall call it *constant-free*.

For the classes $P_{\mathbb{R}}$ and $NP_{\mathbb{R}}$, if we simply restrict the input to be discrete and consider only constant-free BSS machines this results in complexity classes, denoted by $BP(P_{\mathbb{R}}^0)$ and $BP(NP_{\mathbb{R}}^0)$, that may directly be compared to Turing machine based complexity classes. Indeed, it was proved by Allender, Bürgisser, Kjeldgaard-Pedersen and Miltersen [2, Proposition 1.1] that $BP(P_{\mathbb{R}}^0) = P^{\text{PosSLP}}$, where PosSLP is the problem of deciding whether an integer given by a division free arithmetic circuit (i.e. a $\{+, -, *\}$ -circuit using just the constant 1) is positive. While the precise complexity of PosSLP is not known, Allender et al. proved that it is contained in the *counting hierarchy* CH (not to be confused with the consensus halving problem whose abbreviation coincides). The class $BP(NP_{\mathbb{R}}^0)$ is equal to the class $\exists\mathbb{R}$ that was defined by Schaefer and Štefankovič [20] to capture the complexity of the existential theory of the reals ETR. It is known that $NP \subseteq \exists\mathbb{R} \subseteq PSPACE$, where the latter inclusion follows from the decision procedure for ETR due to Canny [7].

We define the class of $\exists\mathbb{R}$ search problems as the following subclass of all real valued search problems. Instances I are encoded as string over a given finite alphabet Σ and we assume there is a polynomial time algorithm that given I computes d_I , where $D_I \subseteq \mathbb{R}^{d_I}$. We next assume that there are polynomial time constant free *BSS machines* that given I and $x \in \mathbb{R}^{d_I}$ checks whether $x \in D_I$, and given I and $x \in D_I$ checks whether $x \in \text{Sol}(I)$. The corresponding language in $\exists\mathbb{R}$ is then $L = \{I \mid \text{Sol}(I) \neq \emptyset\}$.

2.3 Solving real-valued search problems

Let Π be a $\exists\mathbb{R}$ search problem. In analogy with the case of NP search problems, one could consider the task of solving Π to be that of giving as output some member y of $\text{Sol}(I)$ in case $\text{Sol}(I) \neq \emptyset$. In general each member of $\text{Sol}(I)$ may be irrational valued which precludes a Turing machine to compute a solution explicitly. This is in general also the case for a BSS machine, even when allowing machine constants. Regardless, we shall restrict our attention to Turing machines below.

On the other hand, when $\text{Sol}(I) \neq \emptyset$ a solution is guaranteed to exist with coordinates being algebraic numbers, since a member of $\text{Sol}(I)$ may be defined by an existential first-order formula over the reals with only rational-valued coefficients. This means that one could instead compute an indirect description of the coordinates of a solution, for instance by describing isolated roots of univariate polynomials. If such a description could be computed in polynomial time in $|I|$ we could consider that to be a polynomial time solution of Π .

Etessami and Yannakakis [12] suggest several other computational problems one may alternatively consider in place of solving a search problems Π explicitly or exactly. Our main interest is in the problem of *approximation*. We shall assume for simplicity that $D_I \subseteq [-1, 1]^{d_I}$. Together with an instance I of Π we are now given as an auxiliary input a rational number $\varepsilon > 0$, and the task is to compute $x \in \mathbb{Q}^{d_I}$ such that there exist $x^* \in \text{Sol}(I)$ with $\|x^* - x\|_\infty \leq \varepsilon$. We shall turn this into a *discrete* search problem by encoding the coordinates of x as binary strings. More precisely, to Π we shall associate a discrete search problem Π_a where instances are of the form (I, k) , where I is an instance of Π and k is a positive integer. We define $\varepsilon = 2^{-k}$ and let the domain of (I, k) be $D_{I,k} = \{0, 1\}^{d_I(k+3)}$, thereby allowing the specification of a point $x \in D_I$ with coordinates of the form $x_i = a_i 2^{-k+1}$, where $a_i \in \{-2^{k+1}, \dots, 2^{k+1}\}$. The solution set $\text{Sol}(I, k)$ is defined from $\text{Sol}(I)$ by approximating each coordinate. That is, we define $\text{Sol}(I, k) = \{x \in D_{I,k} \mid \exists x^* \in \text{Sol}(I) : \|x^* - x\|_\infty \leq \varepsilon\}$. Note that if we had defined $\text{Sol}(I, k)$ by instead truncating the coordinates of solutions $x^* \in \text{Sol}(I)$ to k bits of precision, we would have obtained the possibly harder problem of *partial computation* which was also considered by Etessami and Yannakakis [12].

We say that Π can be approximated in polynomial time if the approximation problem Π_a can be solved in time polynomial in $|I|$ and k .

2.4 Reductions between search problems

Let Π and Γ be search problems. A *many-one reduction* from Π to Γ consists of a pair of functions (f, g) . The function f is called the instance mapping and the function g the solution mapping. The instance mapping f maps any instance I of Π to an instance $f(I)$ of Γ and for any solution $y \in \text{Sol}(f(I))$ of Γ the solution mapping g maps the pair (I, y) to a solution $x = g(I, y) \in \text{Sol}(I)$ of Π . It is required that $\text{Sol}(f(I)) \neq \emptyset$ whenever $\text{Sol}(I) \neq \emptyset$. We will only consider many-one reductions, and will refer to these simply as *reductions*.

If Π_1 and Π_2 are discrete search problems a reduction (f, g) between Π_1 and Π_2 is a *polynomial time reduction* if both functions f and g are computable in polynomial time. If Π_1 and Π_2 are real-valued search problems it is less obvious which notion of reduction is most appropriate and we shall consider several different types of reductions. For all these we assume that f is computable in polynomial time. The reduction (f, g) is a *real polynomial time reduction* if g is computable in polynomial time by a constant free BSS machine. We shall generally consider this notion of reduction too powerful. In particular the definition does not guarantee that the function g is a continuous function in its second argument y . For this reason we instead consider reductions defined by algebraic circuits over a given basis B of real-valued basis functions.

We say that the reduction (f, g) is a *polynomial time B-circuit reduction* if there is a function computable in polynomial time that maps an instance I to a B -circuit C_I in such a way that C_I computes a function $C_I: D_{f(I)} \rightarrow D_I$ where $g(I, y) = C_I(y)$ for all $y \in \text{Sol}(f(I))$. Note in particular that the size of C_I and the bitsize of all constant gates are bounded by a polynomial in $|I|$. If in addition there exists a constant h such that the depth of C_I is bounded by h for all I we say that the reduction (f, g) is a *polynomial time constant depth B-circuit reduction*. Etessami and Yannakakis [12] defined the even weaker notion where the function

f is a *separable linear* transformation. The reduction (f, g) is an SL-reduction if there is a function $\pi: \{1, \dots, d_I\} \rightarrow \{1, \dots, d_{f(I)}\}$ and rational constants a_i, b_i , for $i = 1, \dots, d_I$, all computable in polynomial time from I , such that for all $y \in \text{Sol}(f(I))$ it holds that $x_i = a_i y_{\pi(i)} + b_i$, where $x = g(I, y)$. Thus an SL-reduction is simply a *projection reduction* together with an individual affine transformation of each coordinate of the solution.

Functions computed by algebraic circuits over the basis $\{+, *\zeta, \max\}$ are piecewise linear. We shall thus call polynomial time $\{+, *\zeta, \max\}$ -circuit reductions for polynomial time piecewise linear reductions, or simply PL-reductions.

It is easy to see that all notions of reductions defined above are transitive, i.e. if Π reduces to Γ and Γ reduces to Λ , then Π reduces to Λ as well.

A desirable property of PL-reductions is that the solution mapping g is *polynomially continuous*. By this we mean that for all rational $\varepsilon > 0$ there is a rational $\delta > 0$ such that for all points x and y of the domain, $\|x - y\|_\infty \leq \delta$ implies $\|g(x) - g(y)\|_\infty \leq \varepsilon$, and the bitsize of δ is bounded by a polynomial in the bitsize of ε and of $|I|$.

2.5 Total real-valued search problems

Like in the case of TFNP where interesting classes of total NP search problems may be defined in terms of existence theorems for finite structures [19, 17], we may define classes of total real valued $\exists\mathbb{R}$ search problems based on existence theorems concerning domains $D_I \subseteq \mathbb{R}^n$. Typical examples of such domains D_I are spheres and balls. Suppose p is either a real number $p \geq 1$ or $p = \infty$. By S_p^n and B_p^n we denote the unit n -sphere and unit n -ball with respect to the ℓ_p -norm defined as $S_p^n = \{x \in \mathbb{R}^{n+1} \mid \|x\|_p = 1\}$ and $B_p^n = \{x \in \mathbb{R}^n \mid \|x\|_p \leq 1\}$, respectively. If p is not specified, we simply assume $p = 2$.

2.5.1 The Brouwer fixed point theorem and FIXP

We recall here the definition of the class FIXP by Etessami and Yannakis [12]. The class FIXP is defined by starting with $\exists\mathbb{R}$ search problems given by the Brouwer fixed point theorem, and afterwards closing the class with respect to SL-reductions. We shall refer to these defining problems as *basic* FIXP problems.

► **Definition 3.** *An $\exists\mathbb{R}$ search problem Π is a basic FIXP problem if every instance I describes a nonempty compact convex domain D_I and a continuous function $F_I: D_I \rightarrow D_I$, computed by an algebraic circuit C_I , and these descriptions must be computable in polynomial time. The solution set is $\text{Sol}(I) = \{x \in D_I \mid F_I(x) = x\}$.*

The Brouwer fixed point theorem guarantees that every basic FIXP problem is a total $\exists\mathbb{R}$ search problem. To define the class FIXP, Etessami and Yannakis restrict attention to a concrete class of basic FIXP problems.

► **Definition 4.** *The class FIXP consists of all total $\exists\mathbb{R}$ search problems that are SL-reducible to a basic FIXP problem for which each domain D_I is a convex polytope described by a set of linear inequalities with rational coefficients and the function F_I is defined by a $\{+, -, *, \div, \max, \min\}$ -circuit C_I .*

The class FIXP_a is the class of strong approximation problems corresponding to FIXP. More precisely, FIXP_a consist of all discrete search problems polynomial time reducible to the problem Π_a for $\Pi \in \text{FIXP}$.

The definition of FIXP is quite robust with respect to the choice of domain and set of basis functions allowed by circuits in the basic FIXP problems. Etessami and Yannakis proved that basic FIXP problems defined by $\{+, -, *, \div, \max, \min, \sqrt{}\}$ -circuits are still in

the class FIXP. Likewise, basic FIXP-problems where D_I is a ball with rational-valued center and diameter, or more generally an ellipsoid given by a rational center-point and a positive-definite matrix with rational entries, are still in the class FIXP [12, Lemma 4.1]. The same argument allows for using as domain the ball B_p^d with respect to the ℓ_p norm for any rational $p \geq 1$ or $p = \infty$, with the coordinates possibly transformed by individual affine functions.

2.5.2 The Borsuk-Ulam theorem, BU, and BBU

A new class BU of total $\exists\mathbb{R}$ search problems based on the Borsuk Ulam theorem was recently introduced by Deligkas et al. [11]. The definition of BU is meant to capture the Borsuk-Ulam theorem as stated in formulation (1) of Theorem 1. Following the definition of FIXP by Etessami and Yannakakis, Deligkas et al. first consider a set of basic search problems and then close the class under reductions. For defining BU, Deligkas et al. restrict their attention to spheres with respect to the ℓ_1 -norm as domains and functions computed by $\{+, -, *, \max, \min\}$ -circuits. Compared to the definition of FIXP, division gates are thus excluded. Having thus fixed the set of basic BU search problems what remains in order to define BU is to settle on a notion of reductions. In their journal paper, Deligkas et al. [11] suggest using reductions computable by general algebraic circuits including non-continuous comparison gates, whereas in the preceding conference paper [10] they did not precisely define a choice of reductions.

We propose defining BU using a different notion of reduction below. We additionally define a class BBU based on the Borsuk-Ulam theorem corresponding to formulation (3) of Theorem 1. We start by defining basic BU and basic BBU problems. We shall restrict our attention to domains being the unit n -sphere and unit n -ball, but with regards to any ℓ_p -norm for $p \geq 1$ or $p = \infty$.

► **Definition 5.**

1. An $\exists\mathbb{R}$ search problem Π is a basic ℓ_p -BU problem if for every instance I we have $D_I = S_p^{d_I}$ and I describes a continuous function $F_I: D_I \rightarrow \mathbb{R}^{d_I-1}$, computed by an algebraic circuit C_I whose description is computable in polynomial time. The solution set is $\text{Sol}(I) = \{x \in D_I \mid F_I(x) = F_I(-x)\}$.
2. An $\exists\mathbb{R}$ search problem Π is a basic ℓ_p -BBU problem if for every instance I we have $D_I = B_p^{d_I}$ and I describes a continuous function $F_I: D_I \rightarrow \mathbb{R}^{d_I}$, which is odd on the boundary $\partial B_p^{d_I}$. The function F_I must be computed by an algebraic circuit C_I whose description is computable in polynomial time. The solution set is $\text{Sol}(I) = \{x \in D_I \mid F_I(x) = 0\}$.

The condition that the function F_I is odd on $\partial B_p^{d_I}$ for basic ℓ_p -BBU problems is a semantic condition. However, typically the function F_I would be defined from a basic ℓ_p -BU problem by a transformation done in a similar way as in the proof of Theorem 1, and thereby F_I would satisfy the condition automatically.

To define the classes BU and BBU, we restrict our attention to domains with respect to the ℓ_∞ -norm.

► **Definition 6.** *The class BU (respectively, BBU) consists of all total $\exists\mathbb{R}$ search problems that are PL-reducible to a basic ℓ_∞ -BU problem (respectively, basic ℓ_∞ -BBU problem) for which the function F_I is defined by a $\{+, -, *, \div, \max, \min\}$ -circuit C_I .*

While the definition of BU in [11] was using as domain the unit sphere with respect to the ℓ_1 -norm and not allowing for division gates, we show in the full version of this paper that these changes do not change the class. We propose choosing PL-reductions for closing the

class under reductions. PL-reductions are sufficient for obtaining all of our results and they are polynomially continuous. Another reason for this choice is that if we restrict the circuits defining the classes FIXP and BU to also be piecewise linear, i.e. be $\{+, * \zeta, \max\}$ -circuits, we obtain the classes LinearFIXP and LinearBU, that when closed under polynomial-time reductions are equal to PPA and PPA, respectively [12, 11].

2.6 Consensus Halving

We give here a formal definition of consensus halving with additive measures as real valued search problems.

► **Definition 7.** *The problem CH is defined as follows. An instance I consists of a list of $\{+, -, *, \div, \max, \min\}$ -circuits C_1, \dots, C_n computing distribution functions F_1, \dots, F_n defined on the interval $A = [0, 1]$. The domain is $D_I = S_1^n$ and $\text{Sol}(I)$ consists of all x for which*

$$\sum_{j: x_j > 0} F_i(t_j) - F_i(t_{j-1}) = \sum_{j: x_j < 0} F_i(t_j) - F_i(t_{j-1}) \quad , \quad (1)$$

where $t_0 = 0$ and $t_j = \sum_{k \leq j} |x_k|$, for $j = 1, \dots, n + 1$.

Given $\{+, -, *, \div, \max, \min\}$ -circuits computing the distribution functions F_i , the function F computing the left-hand-side of equation (1) may now clearly be computed by $\{+, -, *, \div, \max, \min\}$ -circuits as well. The result of Deligkas et al. that CH is contained in BU follows.

The existence proof of a consensus halving by Simmons and Su as well the formulation as an $\exists \mathbb{R}$ search problem by Deligkas et al. match the Borsuk-Ulam theorem as stated in formulation (1) of Theorem 1.

3 Consensus Halving

In this section we present the proof of our main result Theorem 2. This result enables an additional structural result about the class of strong approximation problems $\text{BU}_a = \text{BBU}_a$, showing that the class is unchanged even when allowing root operations as basic operations. We refer to the full version of this paper for details of this.

Suppose we are given a basic $\ell_\infty - \text{BBU}_a$ problem Π_a with circuits over the basis $\{+, -, *, \max, \min\}$. Let (I, k) denote an instance of Π_a and put $\varepsilon = 2^{-k}$. We may in polynomial time compute a circuit C defining a function $F: B_\infty^n \rightarrow \mathbb{R}^n$ that is odd on the boundary S_∞^{n-1} such that $\text{Sol}(I) = \{x \in B_\infty^n \mid F(x) = 0\}$. We now provide a reduction from Π_a to a CH_a -problem. In the reduction we will make use of the “almost implies near” paradigm as expressed in the following lemma. The simple proof is given in the full version of this paper.

► **Lemma 8.** *Let $F: B_\infty^n \rightarrow \mathbb{R}^n$ be a continuous map. For any $\varepsilon > 0$ there is a $\delta > 0$ such that if $\|F(x)\|_\infty \leq \delta$ then there is an $x^* \in B_\infty^n$ such that $\|x - x^*\|_\infty \leq \varepsilon$ and $F(x^*) = 0$.*

The lemma says that for any $\varepsilon > 0$, if $\|F(x)\|_\infty$ is sufficiently close to being zero, then x is ε -close to a real zero of F . When F is computed by an algebraic circuit of polynomial size, it follows by using tools from real algebraic complexity [3] that there exists some fixed polynomial q with integer coefficients such that the above lemma holds true for some $\delta \geq \varepsilon^{2^{q(I)}}$. We refer to the full version of this paper for details. The lemma then holds

true for $\delta = \varepsilon^{2^{q(|I|)}}$, and we may construct this number using a circuit of polynomial size by repeatedly squaring the number ε exactly $q(|I|)$ times. This number will be used by the feedback agents in our CH_a instance in order to ensure that any solution gives a solution to the $\ell_\infty - \text{BBU}_a$ instance.

3.1 Overview of the Reduction

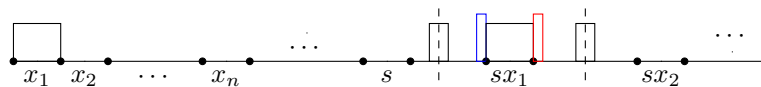
Overview. As in previous works, we describe a consensus halving instance on an interval $A = [0, M]$, where M is bounded by a polynomial in $|I|$, rather than the interval $[0, 1]$. This instance may then be translated to an instance on the interval $[0, 1]$ by simple scaling. Like [16], in the leftmost end of the instance we place the *Coordinate-Encoding* region consisting of n intervals. In a solution these intervals will encode a value $x \in [-1, 1]^n$. A *circuit simulator* C will simulate the circuit of F on this value x . The circuit simulators will consist of a number of agents each implementing one gate of the circuit. However, such a circuit simulator may fail in simulating F properly, so we will use a polynomial number of circuit simulators $C_1, \dots, C_{p(n)}$. Each of these circuit simulators will output n values $[C_j(x)]_1, \dots, [C_j(x)]_n$ into intervals I_{1j}, \dots, I_{nj} immediately after the simulation. Finally, we introduce the so-called *feedback agents* f_1, \dots, f_n . The agent f_i will have some very thin *Dirac blocks* centered in each of the intervals I_{ij} where $j \in [p(n)]$. These agents will ensure that if z is an exact solution to the CH instance, then the encoded value x satisfies that $\|F(x)\|_\infty$ is sufficiently small that we may conclude that x is ε -close to a zero x^* of F .

Label Encoding. For a unit interval I we let I^\pm denote the subsets of I assigned the corresponding label. We define the *label encoding* of I to be a value in $[-1, 1]$ given by the formula $v_l(I) := \mu(I^+) - \mu(I^-)$. This makes sense as I^\pm is measurable, because it is the union of a finite number of intervals.

Coordinate-Encoding Region. The interval $[0, n]$ is called the *Coordinate-Encoding* region. For every $i \in [n]$, the subinterval $[i - 1, i]$ of the Coordinate-Encoding region encodes a value $x_i := v_l([i - 1, i])$ via the label encoding.

Position Encoding. For an interval I which contains only a single cut, thus dividing I into two subintervals $I = I_a \cup I_b$, where I_a is the subinterval at the left of the cut and I_b the subinterval at the right of the cut, we define the *position encoding* of I to be the value $v_p(I) := \mu(I_a) - \mu(I_b)$. We note that $v_p(I) = v_l(I)$ if the labeling sequence is $+/-$, and $v_p(I) = -v_l(I)$ in the case the labeling sequence is $-/+$.

From Label to Position. Before a circuit simulator there is a *sign detection* interval I_s which detects the labeling sequence. Unless it contains a stray cut, this interval will encode a sign $s = \pm 1$ (to be precise 1 if the label is $+$ and -1 if the label is $-$). By placing agents that flip the label as indicated in the figure below, we may now obtain position encodings of the values sx_1, \dots, sx_n . These values will be read-in as inputs to the subsequent circuit simulator.



Circuit Simulators. As mentioned above, each circuit simulator C_j will read-in the values $s_j x_1, \dots, s_j x_n$ and simulate the circuit computing F on this input. They then output their values into n intervals immediately after the simulation.

Feedback Agents. By the discussion after the statement of Lemma 8 we may by repeated squaring construct a circuit of polynomial size in $|I|$ computing a tiny number $\delta > 0$ such that if $\|F(x)\|_\infty \leq \delta$ then x is $(\varepsilon/2)$ -close to a zero of F . Now fix $i \in [n]$ and let c_{ij} denote the centre of the feedback interval I_{ij} that outputs the value $[C_j(s_j \cdot x)]_i$. We then define the i th feedback agent to have constant density $1/\delta$ in the intervals $[c_{ij} - \delta/2, c_{ij} + \delta/2]$.

The reason for having the feedback agents have these very narrow Dirac blocks is that if $F_i(x) > \delta$ for some i , then in any of the “uncorrupted” circuits (i.e. circuits outputting the correct values) all the density of the i th agent will contribute to the same label. Moreover, we will show using the boundary condition of F that the contribution is to the same label in all the uncorrupted circuit simulators. This will contradict that the feedback agents should value I^+ and I^- equally. That is, the feedback agents ensure that $\|F(x)\|_\infty \leq \delta$ if x is the value encoded by an exact solution to the consensus halving instance we construct.

Stray Cuts. Any of the agents implementing one of the gates in a circuit simulator will force a cut to be placed in an interval in that same circuit simulator. The only agents whose cuts we have no control over are the n feedback agents. The expectation is that these agents should make cuts in the Coordinate-Encoding region that flip the label. If they do not do this we will call it a *stray cut*. If a circuit simulator contains a stray cut, we will say nothing about its value.

► **Observation 9.** *If it is not the case that every unit interval encoding a coordinate x_i in the Coordinate-Encoding region contains a cut that flips the label, then the encoded point $x \in B_\infty^n$ will lie on the boundary S_∞^n . With this in mind we may ensure that $x \in S_\infty^n$ or $s_1 = s_2 = \dots = s_{p(n)} = \pm 1$ where the sign is the same as the label of the first interval. This can be done by, if necessary, placing one single-block agent after the Coordinate-Encoding region and each of the circuit simulators (if placing such an agent is necessary depends on, respectively, the number of variables n and the size of the circuits).*

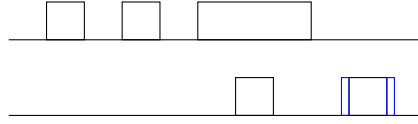
3.2 Construction of Gates

In this section we describe how to construct Consensus-Halving agents implementing the required gates $\{+, -, *, \max, \min\}$ for building the circuit simulators. By placing single-block agents between intervals, we may assume that the labeling sequence is the same in all the intervals of a circuit simulator. First, we show that we may transform the given circuit such that all gates only take values in the interval $[-1, 1]$ on input from B_∞^n .

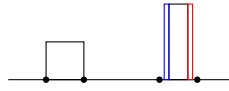
Transforming the Circuit. By propagating every gate to the top of the circuit we may assume that the circuit is layered. Let C' denote the resulting circuit. By repeated squaring we may maintain a gate with value $1/2^{2^d}$ in the d th layer. Suppose $g = \alpha(g_1, g_2)$ is a gate with inputs g_1, g_2 in layer d . We modify the gates as follows: if $\alpha \in \{+, -, \max, \min\}$ then we multiply g_i by $1/2^{2^d}$ before applying α ; if $\alpha = *$, then we multiply the input by 1 before applying α . Finally, we transform C' into the circuit C'' as follows: on input x , the circuit C'' multiplies the input by $1/2$ and then evaluates C' on input $x/2$. Inductively, one may show that if g is a gate in layer d in the circuit C' , then the corresponding gate in the circuit C'' has value $g/2^{2^d}$. As all the gates are among $\{+, -, *, \max, \min\}$, this ensures that all the gates in C'' take values in $[-1, 1]$.

24:16 Strong Approximate Consensus Halving and the Borsuk-Ulam Theorem

Addition Gate $[G_+]$. We may construct an addition gate using two agents. The first agent has two unit input intervals that we assume contain one cut each. This then forces a cut in the long output interval that has length 3. The second agent then truncates this value (a cut is forced by the adjacent narrow rectangles).



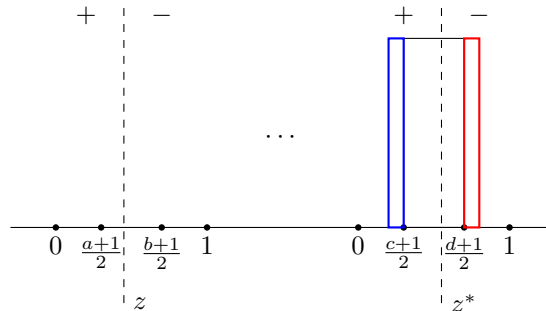
Constant Gate $[G_\zeta]$. Let $\zeta \in [-1, 1] \cap \mathbb{Q}$ be a rational constant. The agent will have a block of unit height in the sign interval and a block of width $\zeta/2$ and height $2/\zeta$ centered in another interval.



Before proceeding with the remaining gates, we construct a general function gate, an agent that implements any decreasing function.

Function Gate $[G_h]$. Let $-1 \leq a < b \leq 1$ and $-1 \leq c < d \leq 1$ be rational numbers and consider a continuously differentiable map $h: [a, b] \rightarrow [c, d]$ satisfying $h(a) = d$ and $h(b) = c$. Let \bar{h} denote the extension of h that is constant on $[-1, a]$ and $[b, 1]$. We now construct an agent with input interval I and output interval O computing this map, that is the agent should force a cut in the output interval such that $\bar{h}(v_p(I)) = v_p(O)$.

The agent that we construct has a block of height $2/(d - c)$ in the sub-interval $[(c + 1)/2, (d + 1)/2]$ of the output interval and density $f(z) := -2h'(2z - 1)/(d - c)$ in the sub-interval $((a + 1)/2, (b + 1)/2)$ of the input interval. We note that f is positive in this interval as h is assumed to be a decreasing map, so it makes sense for the agent to have density f . One may verify that the agent values the input interval and output interval equally. We further add two narrow rectangles adjacent to the output interval. These will ensure that if the cut in the input interval is placed at $z \leq (a + 1)/2$ such that $v_p(I) \leq a$, then the cut in the output interval must be placed at $z^* = (d + 1)/2$, meaning that $v_p(O) = d$. Similarly, if $v_p(I) \geq b$ then $v_p(O) = c$.



Suppose cuts are placed in z in the input interval and in z^* in the output interval. As the agent must value the parts with positive and negative label equally, we get the equality

$$1 = \int_{(a+1)/2}^z \frac{-2h'(2t-1)}{d-c} dt + (z^* - \frac{c+1}{2}) \frac{2}{d-c}$$

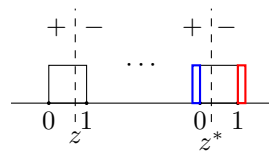
From this we obtain that

$$d - c = - \int_a^{2z-1} h'(u) du + 2z^* - c - 1 = -h(2z - 1) + d + 2z^* - c - 1$$

where we use that $h(a) = d$ by assumption. We conclude that $h(2z - 1) = 2z^* - 1$, that is we obtain the equality $h(v_p(I)) = v_p(O)$.

Using this general function gate, we may now build up the remaining gates required by the circuit.

Multiplication By -1 Gate $[G_{-(\cdot)}]$. In order to realise this gate, we consider the function $h: [-1, 1] \rightarrow [-1, 1]$ given by $x \mapsto -x$. The agent's density function in the input interval is then given by $f(z) = 1$.



Subtraction Gate $[G_-]$. We may build this using the gates $G_{-(\cdot)}$ and G_+ .

Multiplication by $\zeta \in [-1, 1]$ $[G_{\cdot\zeta}]$. If $\zeta < 0$ we may construct $G_{\cdot\zeta}$ as a function gate using the function $h: [-1, 1] \rightarrow [\zeta, -\zeta]$. If $\zeta > 0$ we construct using $-\zeta$ and a minus gate, i.e. $G_{\cdot\zeta} = -G_{(-\zeta)}$.

Maximum Gate $[G_{\max}]$. First we show how to construct a gate computing the absolute value of the input. We may construct gates G_1, G_2 such that $G_1(x) = -\max(x, 0)$ and $G_2(x) = \max(-x, 0)$ as function gates by using the functions $h_1: [0, 1] \rightarrow [-1, 0]$ given by $x \mapsto -x$ and $h_2: [-1, 0] \rightarrow [0, 1]$ given by $x \mapsto -x$. Now, we may construct the absolute value gate as $G_{|\cdot|} = -G_1 + G_2$. We may now construct G_{\max} by using the formula $\max(x, y) = (x + y + |x - y|)/2$.

Minimum Gate $[G_{\min}]$. We may build this using $\min(x, y) = x + y - \max(x, y)$.

Multiplication Gate $[G_{\cdot^2}]$. We start off by constructing a gate squaring the input. First we construct G_1 and G_2 as function gates with respect to $h_1: [-1, 0] \rightarrow [0, 1]$ given by $x \mapsto x^2$ and $h_2: [0, 1] \rightarrow [-1, 0]$ given by $x \mapsto -x^2$. Then we may construct the squaring gate as $G_{(\cdot)^2} = G_1 - G_2$. Now we may use the previously constructed gates to make a multiplication gate via the identity $xy = ((x + y)^2 - x^2 - y^2)/2$.

3.3 Describing valuation functions as circuits

In the description above, we described the valuations of the agents by providing formulas for their densities. However, an instance of CH actually consists of a list of algebraic circuits computing the distribution functions of the agents. In order to construct gates, it is sufficient for agents to have densities that are piece-wise polynomial. Therefore, consider an agent with polynomial densities f_i in the intervals $[a_i, b_i]$ for $i = 1, \dots, s$, and let F_i denote the indefinite integral of f_i . We note that F_i is a polynomial so it may be computed by an algebraic circuit. Now we claim that the distribution function of this agent may be computed by an algebraic circuit via the formula $F(x) = \sum_{i=1}^s [F_i(\max(a_i, \min(x, b_i))) - F_i(a_i)]$.

This is the case, because the summands will be equal to $F_i(a_i) - F_i(a_i) = 0$ if $x < a_i$, to $F_i(x) - F_i(a)$ if $a_i \leq x \leq b_i$ and to $F_i(b) - F_i(a)$ if $x > b_i$, meaning that this formula does indeed calculate the valuation of the agent in the interval $[0, x]$.

3.4 Reduction and Correctness

Recall that we are given an instance (F, ε) of the BBU_a problem and that we have to construct an instance of the CH_a problem. The reduction now outputs an instance of the CH_a problem where the consensus halving instance is constructed as above with $p(n) = 2n + 1$ circuit simulators and the approximation parameter is given by $\varepsilon' = \varepsilon/(4n)$. Let z denote a solution to this CH_a instance. By definition, there exists an exact solution z^* to the consensus-halving problem such that $\|z - z^*\|_\infty \leq \varepsilon'$.

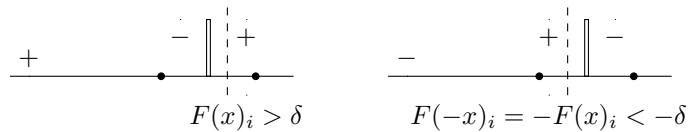
Let x and x^* denote the values encoded by respectively z and z^* in the Coordinate-Encoding region. Suppose, generally, we are given an interval I with a number of cut points t_1, \dots, t_s . Moving a cut point by a distance $\leq \varepsilon'$ we create a new interval I' . This changes the label encoding by at most $2\varepsilon'$, that is $|v_l(I) - v_l(I')| \leq 2\varepsilon'$. Successively, if we move all the cuts by a distance $\leq \varepsilon'$, then we get an interval I^* such that $|v_l(I) - v_l(I^*)| \leq 2s\varepsilon'$. As $\|z - z^*\|_\infty \leq \varepsilon'$ and any of the subintervals in the Coordinate-encoding region can contain at most n cuts, we conclude that $\|x - x^*\|_\infty \leq 2n\varepsilon' = 2n(\varepsilon/(4n)) = \varepsilon/2$. In order to show that x is ε -close to a zero of F , it now suffices by the triangle inequality to show that x^* is $(\varepsilon/2)$ -close to a zero of F . This will follow from the two following lemmas.

► **Lemma 10.** *If there are no stray cuts in the exact solution z^* , then the associated value x^* encoded in the Coordinate-encoding region satisfies $F(x^*) = 0$.*

Proof. We recall that if the solution z^* contain no stray cuts, then the signs of all the circuit simulators are equal $s_1 = \dots = s_{2n+1} = s$ where $s = \pm 1$. Furthermore, all the circuit simulators will output the same values $F_1(sx^*), \dots, F_n(sx^*)$ into the feedback intervals. Thus, there can be no cancellation, so in order for the feedback agents to value the positive and negative part equally it must be the case that $F(sx^*) = 0$. ◀

► **Lemma 11.** *If there is a stray cut in the exact solution z^* , then the associated value x^* encoded in the Encoding-region satisfies the inequality $\|F(x^*)\|_\infty \leq \delta$.*

Proof. Suppose toward contradiction that $|F(x^*)_i| > \delta$ for some i . Without loss of generality we assume that $F(x^*)_i > \delta$. As there is a stray cut, the Coordinate-Encoding region can contain at most $n - 1$ cuts. Thus, at least one of the coordinates x_i^* must be ± 1 showing that $x^* \in S^{n-1}$. From this and the boundary condition we conclude that $F(x^*) = -F(-x^*)$. Furthermore, there is at most n stray cuts, so at most n circuit simulators can become corrupted. This means that $n + 1$ circuit simulators work correctly. Now suppose that the circuit simulator C_j is uncorrupted. If the label is $s_j = +1$, then C_j will output $F(x^*)$ into the feedback region and the labeling sequence will be $+/-$; if the label is $s_j = -1$ then C_j will output $F(-x^*) = -F(x^*)$ into the feedback region and the labeling sequence will be $-/+$. This is indicated below:



From this we conclude that the $n + 1$ uncorrupted circuit simulators altogether contribute $(n + 1)\delta$ to the part with negative label. However, the n corrupted circuit simulators can contribute at most $n\delta$ to the part with positive label. This implies that f_i cannot value the negative and positive part equally. This contradicts the assumption that z^* is an exact consensus-halving. We conclude that $\|F(x^*)\|_\infty \leq \delta$. ◀

By the two lemmas above, it follows that the value x^* encoded by the exact consensus-halving z^* satisfies the inequality $\|F(x^*)\|_\infty \leq \delta$. By choice of δ , this implies that there exists some x^{**} such that $\|x^* - x^{**}\|_\infty \leq \varepsilon/2$ and $F(x^{**}) = 0$. From the discussion before the two lemmas, it follows that x is ε -close to a zero of F and is thus a solution to the BBU_a instance (F, ε) .

Mapping back a Solution. What remains is to show that we may recover a solution x to the BBU_a instance from the solution z to the CH_a instance. Recall that in a solution $z = (z_1, \dots, z_N)$ to the consensus-halving problem $|z_i|$ and $\text{sgn}(z_i)$ represents the length and label of the i th interval. For $i \leq n$ and $j \leq n + 1$ we let $t_j = \sum_{k=1}^{j-1} |z_k|$ and define

$$\begin{aligned} x_{ij}^+ &= \max(0, \min(t_{j-1} + z_j, i) - \max(t_{j-1}, i - 1)) \quad \text{and} \\ x_{ij}^- &= \max(0, \min(t_{j-1} - z_j, i) - \max(t_{j-1}, i - 1)) \end{aligned}$$

These numbers may be computed efficiently by a circuit over the basis $\{+, -, \max, \min\}$. We notice that if $z_j > 0$ then $x_{ij}^- = 0$ (and if $z_j < 0$ then $x_{ij}^+ = 0$). Furthermore, by checking a couple of cases, one finds that if $z_j > 0$ (respectively $z_j < 0$) then x_{ij}^+ (respectively x_{ij}^-) is the length of the j th interval that is contained in $[i - 1, i]$. As the coordinate-encoding region can contain at most n cuts (corresponding to at most $n + 1$ intervals), we deduce from the above that the values encoded can be computed as $x_i = \sum_{j=1}^{n+1} x_{ij}^+ - x_{ij}^-$, for every $i \leq n$. If there is a stray cut then both x and $-x$ are valid solutions by the boundary condition of F . If there is no stray cut, then $s_1 = s_2 = \dots = s_{p(n)} = s = \text{sgn}(z_1)$ by Observation 9 and in this case we may recover a solution as sx .

References

- 1 James Aisenberg, Maria Luisa Bonet, and Sam Buss. 2-d tucker is PPA complete. *J. Comput. Syst. Sci.*, 108:92–103, 2020. doi:10.1016/j.jcss.2019.09.002.
- 2 Eric Allender, Peter Bürgisser, Johan Kjeldgaard-Pedersen, and Peter Bro Miltersen. On the complexity of numerical analysis. *SIAM J. Comput.*, 38(5):1987–2006, 2009. doi:10.1137/070697926.
- 3 S. Basu, R. Pollack, and M. Roy. *Algorithms in Real Algebraic Geometry*. Springer, second edition, 2008.
- 4 Lenore Blum, M. Schub, and Steve Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bull. Amer. Math. Soc.*, 21:1–46, July 1989. doi:10.1090/S0273-0979-1989-15750-9.
- 5 Karol Borsuk. Drei sätze über die n-dimensionale euklidische sphäre. *Fundamenta Mathematicae*, 20(1):177–190, 1933. doi:10.4064/fm-20-1-177-190.
- 6 L. E. J. Brouwer. Über abbildung von mannigfaltigkeiten. *Mathematische Annalen*, 71:97–115, 1911. doi:10.1007/BF01456931.
- 7 John Canny. Some algebraic and geometric computations in PSPACE. In *STOC*, pages 460–467. ACM, January 1988. doi:10.1145/62212.62257.
- 8 Xi Chen and Xiaotie Deng. Settling the complexity of two-player Nash equilibrium. In *FOCS 2006*, pages 261–272. IEEE Computer Society Press, 2006.

- 9 Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.
- 10 Argyrios Deligkas, John Fearnley, Themistoklis Melissourgos, and Paul G. Spirakis. Computing exact solutions of consensus halving and the Borsuk-Ulam theorem. In *ICALP*, volume 132 of *LIPICs*, pages 138:1–138:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ICALP.2019.138.
- 11 Argyrios Deligkas, John Fearnley, Themistoklis Melissourgos, and Paul G. Spirakis. Computing exact solutions of consensus halving and the Borsuk-Ulam theorem. *Journal of Computer and System Sciences*, 117:75–98, 2021. doi:10.1016/j.jcss.2020.10.006.
- 12 Kousha Etesami and Mihalis Yannakakis. On the complexity of Nash equilibria and other fixed points. *SIAM J. Comput.*, 39(6):2531–2597, 2010.
- 13 Aris Filos-Ratsikas, Søren Kristoffer Stiil Frederiksen, Paul W. Goldberg, and Jie Zhang. Hardness results for consensus-halving. In *MFCs*, volume 117 of *LIPICs*, pages 24:1–24:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.MFCs.2018.24.
- 14 Aris Filos-Ratsikas and Paul W. Goldberg. Consensus halving is PPA-complete. In *STOC*, pages 51–64. ACM, 2018. doi:10.1145/3188745.3188880.
- 15 Aris Filos-Ratsikas and Paul W. Goldberg. The complexity of splitting necklaces and bisecting ham sandwiches. In *STOC*, pages 638–649. ACM, 2019. doi:10.1145/3313276.3316334.
- 16 Aris Filos-Ratsikas, Alexandros Hollender, Katerina Sotiraki, and Manolis Zampetakis. Consensus-halving: Does it ever get easier? In *EC*, pages 381–399. ACM, 2020. doi:10.1145/3391403.3399527.
- 17 Paul W. Goldberg and Christos H. Papadimitriou. Towards a unified complexity theory of total functions. *Journal of Computer and System Sciences*, 94:167–192, 2018. doi:10.1016/j.jcss.2017.12.003.
- 18 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988.
- 19 Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994.
- 20 Marcus Schaefer and Daniel Štefankovič. Fixed points, Nash equilibria, and the existential theory of the reals. *Theory Comput Syst*, 60:172–193, November 2017. doi:10.1007/s00224-015-9662-0.
- 21 Forest W. Simmons and Francis Edward Su. Consensus-halving via theorems of Borsuk-Ulam and Tucker. *Math. Soc. Sci.*, 45(1):15–25, 2003. doi:10.1016/S0165-4896(02)00087-2.
- 22 Francis Edward Su. Borsuk-Ulam implies Brouwer: A direct construction. *The American Mathematical Monthly*, 104(9):855–859, 1997. doi:10.2307/2975293.
- 23 Sergey P. Tarasov and Mikhail N. Vyalyi. Semidefinite programming and arithmetic circuit evaluation. *Discrete Applied Mathematics*, 156(11):2070–2078, 2008. doi:10.1016/j.dam.2007.04.023.
- 24 Alexey Yu. Volovikov. Borsuk-Ulam implies Brouwer: A direct construction revisited. *Am. Math. Mon.*, 115(6):553–556, 2008.