PURE-CIRCUIT:
Strong Inapproximability for PPAD

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

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

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

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

Abstract

The current state-of-the-art methods for showing inapproximability in PPAD arise from the ε-Generalized-Circuit (ε-GCIRCUIT) problem. Rubinstein (2018) showed that there exists a small unknown constant ε for which ε-GCIRCUIT is PPAD-hard, and subsequent work has shown hardness results for other problems in PPAD by using ε-GCIRCUIT as an intermediate problem.

We introduce PURE-CIRCUIT, a new intermediate problem for PPAD, which can be thought of as ε-GCIRCUIT pushed to the limit as $\varepsilon \to 1$, and we show that the problem is PPAD-complete. We then prove that ε-GCIRCUIT is PPAD-hard for all $\varepsilon < 0.1$ by a reduction from PURE-CIRCUIT, and thus strengthen all prior work that has used GCIRCUIT as an intermediate problem from the existential-constant regime to the large-constant regime.

We show that stronger inapproximability results can be derived by reducing directly from PURE-CIRCUIT. In particular, we prove tight inapproximability results for computing ε-well-supported Nash equilibria in two-action polymatrix games, as well as for finding approximate equilibria in threshold games.

1 Introduction

The complexity class PPAD has played a central role in determining the computational complexity of many problems arising in game theory and economics [Pap94]. The celebrated results of Daskalakis, Goldberg, and Papadimitriou [DGP09] and Chen, Deng, and Teng [CDT09] established that finding a Nash equilibrium in a strategic form game is PPAD-complete, and subsequent to this breakthrough many other PPAD-completeness results have been shown [CSVY08, CDDT09, VY11, DQS12, Das13, KPR+13, CDO15, CPY17, SSB17, Meh18, Rub18, DFS20, CKK21a, CKK21b, PP21, GH21, FRGH+21, DSZ21, CCPY22].

These celebrated results not only showed that it is PPAD-hard to find an exact equilibrium, but also that finding approximate solutions is PPAD-hard. The result of Daskalakis, Goldberg, and Papadimitriou [DGP09] showed that finding an ε-Nash equilibrium is PPAD-complete when ε is exponentially small, while the result of Chen, Deng, and Teng [CDT09] improved this to show hardness for polynomially small ε. This lower bound is strong enough to rule out the existence of an FPTAS for the problem.

The main open question following these results was whether equilibrium computation problems in PPAD were hard for *constant* ε, which would also rule out the existence of a PTAS. Here one must be careful, because some problems do in fact admit approximation schemes. For example, in the case of two-player strategic-form games, a quasipolynomial-time approximation scheme is known [LMM03], meaning that the problem cannot be hard for a constant ε unless every problem in PPAD can be solved in quasipolynomial time. But for other types of game such results are not known. This includes *polymatrix games*, which are n-player games with succinct representation [Jan68].

In another breakthrough result, Rubinstein [Rub18] developed techniques for showing constant inapproximability within PPAD, by proving that there exists a constant ε such that finding an ε-well-supported Nash equilibrium in a polymatrix game is PPAD-complete. This lower bound is obtained by first showing constant inapproximability for the ε-*Generalized-Circuit* (ε-GCircuit) problem introduced by Chen, Deng, and Teng [CDT09], and then utilizing the known reduction from GCircuit to polymatrix games [DGP09].

Rubinstein's lower bound has since been used to show constant inapproximability for other problems. Rubinstein himself showed constant inapproximability for finding Bayesian Nash equilibria, relative approximate Nash equilibria, and approximate Nash equilibria in non-monotone markets [Rub18]. Subsequent work has shown constant inapproximability for finding clearing payments in financial networks with credit default swaps [SSB17], finding equilibria in first-price auctions with subjective priors [FRGH+21], finding throttling equilibria in auction markets [CKK21b], finding equilibria in public goods games on directed networks [PP21], and finding consensus-halving solutions in fair division [FRFGZ18, GHI+22].

Rubinstein's lower bound is an *existential-constant* result, meaning that it shows that there exists some constant ε below which the problem becomes PPAD-hard. The fact that such a constant exists is important, since it rules out a PTAS. On the other hand, Rubinstein does not give any concrete lower bound on the size of the constant (understandably so, since this was not the purpose of his work). One could of course deduce such a lower bound by a careful examination of his reduction, but it is clear that this would yield an extremely small constant. Due to this, all of the other results that have utilized Rubinstein's lower bound are likewise existential-constant results, which rule out PTASs but do not give any concrete lower bounds.

Ultimately, this means that existing work does not rule out an efficient algorithm that finds, say, a 0.001-approximate solution for any of these problems, which would likely be more than enough for most practical needs. Arguably, a player would be more than happy to know that her strategy is an optimal best-response, up to a loss of at most 0.001 in her utility value. Moreover, the existing work gives us no clue as to where the threshold for hardness may actually lie. To address these questions one would need to prove a *large-constant* inapproximability result, giving hardness for a known substantial constant.

Rubinstein's lower bound is the ultimate source of all of the recent existential-constant lower bounds, so if one seeks a large-constant lower bound, then Rubinstein's result is the bottleneck. Attempting to directly strengthen or optimize Rubinstein's result does not seem like a promising direction. His proof, while ingenious, is very involved, and does not lend itself to easy optimization. Furthermore, it consists of many moving parts, so that even if one was able to optimize each module, the resulting constant would still be very small.

Our contribution. In this paper we introduce the techniques needed to show large-constant inapproximability results for problems in PPAD. Our key technical innovation is the introduction of a new intermediate problem, called PURE-CIRCUIT, which we show to be PPAD-complete.

Then, by reducing onwards from PURE-CIRCUIT, we show large-constant inapproximability results for a variety of problems in PPAD. In this sense, PURE-CIRCUIT now takes on the role that ε-GCIRCUIT has taken in the past, as an important intermediate problem from which all other results of this type are derived.

The PURE-CIRCUIT problem itself can be thought of as a version of ε-GCIRCUIT that is taken to its limits, and also dramatically simplified. In fact, the problem has only two gates (or, in a different formulation, three gates), which should be compared to ε-GCIRCUIT, which has nine distinct gates. Perhaps more importantly, the gates in PURE-CIRCUIT have very weak constraints on their outputs: the gates can be thought of as taking inputs in $[0, 1]$, and producing outputs in $[0, 1]$, but the gates themselves essentially only care about the values 0 and 1, with all other values being considered to be "bad" or "garbage" values (which we will later simply denote by "$\perp$", instead of using values in $(0, 1)$). This should be compared to ε-GCIRCUIT gates, where, for example, one must output a value in $[0, 1]$ that is within ε of the sum of two inputs.

Combined, these properties make PURE-CIRCUIT a very attractive problem to reduce from when showing a hardness result, since one only has to implement two (or three) gates, and the constraints that one must simulate are very loose, making them easy to implement. We formally introduce PURE-CIRCUIT, and compare it to ε-GCIRCUIT, in Section 2.

Our main result is to show that the PURE-CIRCUIT problem is PPAD-complete. It is worth noting that there is no ε in this result, and in fact the PURE-CIRCUIT problem does not even take a parameter ε in its definition. This is because, in some sense, the PURE-CIRCUIT problem can be viewed as a variant of ε-GCIRCUIT in which we have taken the limit $\varepsilon \to 1$. We give further justification of this idea in Section 2, but at a high level, this means that there is no loss of ε in our main hardness result, with the only losses coming when one reduces onwards from PURE-CIRCUIT. The proof of our main result is presented in Section 3, but we present a brief exposition of the main ideas in Section 1.1.

Finally, in Section 4 we present a number of new large-constant hardness results for problems in PPAD, all of which are shown via reductions from PURE-CIRCUIT. We begin by showing that ε-GCIRCUIT is PPAD-hard for all $\varepsilon < 0.1$, giving a direct strengthening of Rubinstein's lower bound. This also implies large-constant inapproximability results for all of the problems that currently have existential-constant lower bounds proved via GCIRCUIT. However, to determine the constant, one would need to determine the amount of ε that is lost in each of the onward reductions, and these reductions often did not optimize this, since they were proving existential-constant lower bounds.

We argue that the way forward now is providing *direct* reductions from PURE-CIRCUIT in order to get the best possible hardness results. As evidence of this, we present the first *tight* inapproximability result for additive approximate equilibria in polymatrix games. Via a direct reduction from PURE-CIRCUIT, we show that finding an ε-well-supported Nash equilibrium in a polymatrix game (POLYMATRIXWSNE) is PPAD-hard for all $\varepsilon < 1/3$, even when every player only has two actions. This is much stronger than the lower bound of 0.05 that we would have obtained by a reduction from our lower bound for GCIRCUIT. It is also a tight result for two-action games: we give a polynomial-time algorithm for finding a 1/3-well-supported Nash equilibrium, and so our lower bound completely characterizes the computational complexity of approximate well-supported equilibria in two-action polymatrix games. Similarly, we also provide a tight inapproximability result for computing approximate equilibria in threshold games, a problem introduced by Papadimitriou and Peng [PP21].

We summarize the hardness results obtained through a direct reduction from Pure-Circuit in the table below.

Problem	Hardness Threshold
GCircuit	0.1
PolymatrixWSNE	1/3
PolymatrixNE	1/48
ThresholdGameNE	1/6

We note that PolymatrixWSNE and ThresholdGameNE have themselves both been used as intermediate problems for showing other constant inapproximability results in PPAD [Rub18, PP21, CKK21b, CCPY22], and thus our lower bounds potentially strengthen those results too. We provide an example in Appendix C, where we show that computing a relative ε-WSNE in a bimatrix game with non-negative payoffs is PPAD-complete for any $\varepsilon \leq 1/57$, by using an improved version of a reduction from PolymatrixWSNE due to Rubinstein [Rub18].

1.1 Proof Overview for Our Main Result

We begin this proof overview by defining a very weak version of Pure-Circuit. An instance of the problem consists of a Boolean circuit using the standard gates NOT, AND, and OR, but with the following tweak: the circuit is allowed to have *cycles*. A solution to the problem is an assignment of values to each node of the circuit, so that all gates are satisfied. If we are only allowed to assign values in $\{0, 1\}$ to the nodes, then it is easy to see that the problem is not a total search problem, i.e., some instances do not have a solution. For example, there is no way to assign consistent values to a cycle of three consecutive NOT gates.

In order to ensure that the problem is total (and can thus be used to prove PPAD-hardness results), we make the value space continuous by extending it to $[0, 1]$. We extend the definition of the logical gates NOT, AND, and OR to non-Boolean inputs in the most permissive way: if at least one input to the gate is not a pure bit (i.e., not in $\{0, 1\}$), then the gate is allowed to output any value in $[0, 1]$. The attentive reader might observe that this problem is now trivial to solve: just assign arbitrary values in $(0, 1)$ to all the gates.

It is thus clear that the definition of the problem needs to be extended, by adding extra gates or by strengthening existing gates, so that the problem becomes PPAD-hard. However, in order to discover the least amount of additional structure needed to make the problem hard, it is instructive to proceed with this definition for now, and attempt to prove hardness.

In order to prove the PPAD-hardness of the problem, we cannot follow Rubinstein's approach, which goes through the construction of a continuous Brouwer function, because Pure-Circuit only offers very weak gates. Instead, we proceed via a direct reduction from the StrongSperner problem, a discrete problem that is a computational version of Sperner's Lemma. The problem was shown to be PPAD-hard by Daskalakis, Skoulakis, and Zampetakis [DSZ21] (who called it the HighD-BiSperner problem), and is the "PPAD-analogue" of the StrongTucker problem which was recently used to prove PPA-hardness results [DFHM22]. This approach completely bypasses the continuous aspect of all such existing hardness reductions and enables us to work with the very weak gates that Pure-Circuit offers.

At a high level, our hardness construction works as follows: the Pure-Circuit instance implements the evaluation of the StrongSperner labeling on some input point x (represented in unary by multiple nodes) and then uses a feedback mechanism to ensure that the circuit is only satisfied if x is a solution to the StrongSperner instance. The full reduction is presented in Section 3, but we mention here the two main obstacles when trying to implement this idea, and how to overcome them.

1. **The input point x might not be represented by a valid bitstring.** Indeed, since the gates take values in $[0, 1]$ (and values in $(0, 1)$ essentially do not carry any information), there is no guarantee that the input x will be represented by bits $\{0, 1\}$. But then the implementation of the StrongSperner labeling (which is given as a Boolean circuit) will also fail. To resolve this issue, we introduce a new gate, the PURIFY gate, which, on any input, outputs two values, with the guarantee that at least one of them is a "pure" bit, i.e., 0 or 1. If the input is already a pure bit, then both outputs are guaranteed to be copies of the input. Using a binary tree of PURIFY gates, we can now create many copies of x, such that most of them consist only of pure bits, and then use the logical gates to compute the StrongSperner labeling correctly on these good copies.

2. **How to implement the feedback mechanism?** Given the outputs of the StrongSperner labeling at all the copies of x, we now need to provide some kind of feedback to x, so that x is forced to change if it is not a solution of StrongSperner. It turns out that this step can be performed if we have access to *sorting*: given a list of values in $[0, 1]$, sort them from smallest to largest. Unfortunately, this is impossible to achieve with the gates at our disposal, namely standard logical gates and the PURIFY gate. We circumvent this obstacle by observing that: (i) it is sufficient to be able to perform some kind of "weak sorting" (essentially, we only care about pure bits being sorted correctly), and (ii) this weak sorting can be achieved if we make our logical gates *robust*. For example, the robust version of the AND gate outputs 0, whenever at least one of its inputs is 0, irrespective of whether the other input is a pure bit or not.

With these two extensions in hand—namely, the PURIFY gate and the robustness of the logical gates— it is now possible to prove PPAD-hardness of the problem. A very natural question to ask is: Is it really necessary to add both extensions for the problem to be hard? In Appendix A we show that any attempt to weaken the gate-constraints makes the problem polynomial-time solvable. In particular, the introduction of the PURIFY gate is not enough by itself to make the problem PPAD-hard; the robustness of the logical gates is also needed.

The robustness of, say, the AND gate seems like a very natural constraint to impose. It is consistent with the meaning of the logical AND operation, but we also observe in our applications that this "robustness" seems to always be automatically satisfied in all simulations of the AND gate. On the other hand, the PURIFY gate, which might look a bit unnatural or artificial at first, actually corresponds to the simplest possible version of a bit decoder, a crucial tool in all prior works. As mentioned above, we show in Appendix A that these are the minimal gate-constraints that are needed for the problem to be PPAD-hard. In that sense, we argue that Pure-Circuit captures the essence of PPAD-hardness: it consists of the minimal set of ingredients that are needed for a problem to be PPAD-hard.

The attentive reader might have noticed that our gates do not distinguish between different values in $(0, 1)$. For this reason, it will be more convenient to use a single symbol to denote such values in the definition of Pure-Circuit (Section 2) and in the rest of this paper. As explained in more detail in Section 2, the symbol "$\perp$" will be used to denote these "garbage" values. In other words, the nodes of the circuit will take values in $\{0, 1, \perp\}$ instead of $[0, 1]$.

2 The Pure-Circuit Problem

In this section we define our new problem Pure-Circuit and state our main result, namely its PPAD-completeness. Before defining Pure-Circuit, we begin by explaining the intuition behind its definition, and how it relates to the Generalized-Circuit (GCircuit) problem.

The Generalized-Circuit problem. In the Generalized-Circuit (GCircuit) problem (formally defined in Section 4.1) we are given a circuit and the goal is to assign a value to each node of the circuit so that each gate is computed correctly. Importantly, the circuit is a *generalized* circuit, meaning that cycles are allowed. If cycles were not allowed, then it would be easy to find values satisfying all gates: just pick arbitrary values for the input gates, and then evaluate the circuit on those inputs.

Every node of GCircuit must be assigned a value in $[0, 1]$, and the gates are arithmetic gates, such as addition, subtraction, multiplication by a constant (with output truncated to lie in $[0, 1]$), and suitably defined logical gates. Reducing from GCircuit is very useful for obtaining hardness of approximation results, because the problem remains PPAD-hard, even when we allow some error at every gate. In the ε-GCircuit problem, the goal is to assign a value in $[0, 1]$ to each node of the circuit, so that each gate is computed correctly, up to an additive error of $\pm\varepsilon$.

The problem was first defined by Chen et al. [CDT09], who proved that it is PPAD-hard for inverse polynomial ε, and who used it to prove PPAD-hardness of finding Nash equilibria in bimatrix games. Prior to that, Daskalakis et al. [DGP09] had implicitly proved that it is PPAD-hard for inverse exponential ε. Rubinstein's [Rub18] breakthrough result proved that there exists some constant $\varepsilon > 0$ such that ε-GCircuit remains PPAD-hard.

Taking the limit $\varepsilon \to 1$. In order to get strong inapproximability results, it seems necessary to prove hardness of ε-GCircuit for large, explicit, values of ε. Ideally, we would like to obtain hardness for the largest possible ε. While it is unclear what that value is for GCircuit, in theory, as long as $\varepsilon < 1$ the output of a gate still carries some information. Namely a gate whose actual output should be 0 cannot take the value 1.

This observation leads us to define a problem to essentially capture the setting $\varepsilon \to 1$. In that case, a node carries information only if its value is 0 or 1. Otherwise, its value is irrelevant. As a result, the natural operations to consider in this setting are simple Boolean operations, such as NOT, AND, OR, NAND, and NOR. We only require these gates to output the correct result when their input is relevant, i.e., 0 or 1. For example, the NOT gate should output 1 on input 0, and output 0 on input 1, but there is no constraint on its output when the input lies in $(0, 1)$.

Since values in $(0, 1)$ do not carry any information, and are as such interchangeable (e.g., a value $1/2$ can be replaced by $1/3$ without impacting any of the gates), we will instead use the symbol "$\perp$" to denote any and all values in $(0, 1)$. In other words, instead of assigning a value in $[0, 1]$ to each gate, we will assign a value in $\{0, 1, \perp\}$, where $\perp$ is interpreted as a "garbage" value, i.e., not corresponding to a pure bit value 0 or 1. With this new notation, the updated description of the NOT gate would be that it must output 1 on input 0, it must output 0 on input 1, and it can output anything (namely, 0, 1, or $\perp$) on input $\perp$.

Unfortunately, if we only allow these logical gates, then the problem is trivial to solve: assigning the "garbage" value $\perp$ (or any value in $(0, 1)$ if we use the old notation) to every node will satisfy all gates. Thus, we need a gate that makes this impossible.

The PURIFY gate. To achieve this, we introduce the PURIFY gate: a gate with one input and two outputs, which, intuitively, "purifies" its input. When fed with an actual pure bit, the PURIFY gate outputs two copies of the input bit. However, when the input is not a pure bit, the gate still ensures that at least one of its two outputs is a pure bit. In more detail:

- If the input is 0, then both outputs are 0.
- If the input is 1, then both outputs are 1.
- If the input is $\perp$, then at least one of the outputs is a pure bit, i.e., 0 or 1.

Note that the gate is quite "under-defined". For example, we do not specify which pure bit the gate should output when the input is $\perp$, nor do we specify the output on which this bit appears.

This is actually an advantage, because it makes it easier to reduce from the problem, since the less constrained the gates are, the easier it is to simulate them in the target application problem.

Robustness of the logical gates. The introduction of the PURIFY gate makes the problem non-trivial: if a PURIFY gate appears in the circuit, then assigning the "garbage" value $\perp$ to all nodes is no longer a solution. However, it turns out that one more modification is needed to make the problem PPAD-hard: we have to make the logical gates *robust*. For the AND gate, this means the following: if one of its two inputs is 0, then the output is 0, no matter what the other input is (even if it is not a pure bit, i.e., if it is $\perp$). Similarly, for the OR gate we require that the output be 1 when at least one of the two inputs is 1. Robustness is defined analogously for NAND and NOR.

We show that introducing the PURIFY gate and making the logical gates robust is enough to make the problem PPAD-complete. Next, we define the problem formally and state our main result.

Formal definition. In the definition below, we use the PURIFY and NOR gates, because these two gates are enough for the problem to already be PPAD-complete. However, the problem remains hard for various other combinations of gates and restrictions on the interactions between nodes, as we detail in Corollaries 2.2 and 2.3. In Appendix A we discuss the definition in more detail, and explain why any attempt at relaxing the definition (in particular, removing the robustness) makes the problem polynomial-time solvable.

Definition 1 (PURE-CIRCUIT). An instance of PURE-CIRCUIT is given by a vertex set $V = [n]$ and a set G of gate-constraints (or just *gates*). Each gate $g \in G$ is of the form $g = (T, u, v, w)$ where $u, v, w \in V$ are distinct nodes and $T \in \{\mathsf{NOR}, \mathsf{PURIFY}\}$ is the type of the gate, with the following interpretation.

- If $T = \mathsf{NOR}$, then u and v are the inputs of the gate, and w is its output.
- If $T = \mathsf{PURIFY}$, then u is the input of the gate, and v and w are its outputs.

We require that each node is the output of exactly one gate.

A solution to instance (V, G) is an assignment $\mathbf{x} : V \to \{0, 1, \perp\}$ that satisfies all the gates, i.e., for each gate $g = (T, u, v, w) \in G$ we have:

- if $T = \mathsf{NOR}$, then $\mathbf{x}$ satisfies (left: mathematically; right: truth table)

$$\begin{aligned} \mathbf{x}[u] = \mathbf{x}[v] = 0 &\implies \mathbf{x}[w] = 1 \\ (\mathbf{x}[u] = 1) \vee (\mathbf{x}[v] = 1) &\implies \mathbf{x}[w] = 0 \end{aligned}$$

u	v	w
0	0	1
1	$\{0,1,\perp\}$	0
$\{0,1,\perp\}$	1	0
Else		$\{0,1,\perp\}$

- if $T = \mathsf{PURIFY}$, then $\mathbf{x}$ satisfies

$$\begin{aligned} &\{\mathbf{x}[v], \mathbf{x}[w]\} \cap \{0,1\} \neq \emptyset \\ \mathbf{x}[u] \in \{0,1\} &\implies \mathbf{x}[v] = \mathbf{x}[w] = \mathbf{x}[u] \end{aligned}$$

u	v	w
0	0	0
1	1	1
$\perp$	At least one output in $\{0,1\}$	

The following theorem is our main technical result and is proved in Section 3.

Theorem 2.1. *The* PURE-CIRCUIT *problem is* PPAD*-complete.*

The most important part of this statement is of course the PPAD-hardness of Pure-Circuit, but let us briefly discuss the other part, namely the PPAD-membership. This is obtained as a byproduct of our results in Section 4, where we reduce Pure-Circuit to various problems that are known to lie in PPAD. However, there is also a more direct way to prove membership in PPAD, and in particular to establish the existence of a solution, and we briefly sketch it here. Indeed, the Pure-Circuit problem can be reduced to the problem of finding a Brouwer fixed point of a continuous function F, a problem known to lie in PPAD [Pap94, EY10]. Given an instance of Pure-Circuit with n nodes, the function $F : [0,1]^n \to [0,1]^n$ is constructed by letting $x \in [0,1]^n$ represent an assignment of values to the n nodes, and by defining $F_i(x) \in [0,1]$ as a continuous function that outputs a valid value for the ith node, given that the other nodes have values according to assignment x (where any value in $(0,1)$ is interpreted as "$\perp$"). For every type of gate, it is not hard to construct a continuous piecewise-linear function F_i (or, in the case of PURIFY, two such functions F_i and F_j) that satisfies the constraints of that type of gate.

2.1 Alternative Gates and Further Restrictions

In this section, we present various versions of the problem that remain PPAD-complete, in particular versions that use alternative gates and have additional restrictions.

More gates. We define the following additional gates.

- If $T =$ COPY in $g = (T, u, v)$, then $\mathbf{x}$ satisfies

$$\mathbf{x}[u] = 0 \implies \mathbf{x}[v] = 0$$
$$\mathbf{x}[u] = 1 \implies \mathbf{x}[v] = 1$$

u	v
0	0
1	1
$\perp$	$\{0,1,\perp\}$

- If $T =$ NOT in $g = (T, u, v)$, then $\mathbf{x}$ satisfies

$$\mathbf{x}[u] = 0 \implies \mathbf{x}[v] = 1$$
$$\mathbf{x}[u] = 1 \implies \mathbf{x}[v] = 0$$

u	v
0	1
1	0
$\perp$	$\{0,1,\perp\}$

- If $T =$ OR in $g = (T, u, v, w)$, then $\mathbf{x}$ satisfies

$$\mathbf{x}[u] = \mathbf{x}[v] = 0 \implies \mathbf{x}[w] = 0$$
$$(\mathbf{x}[u] = 1) \vee (\mathbf{x}[v] = 1) \implies \mathbf{x}[w] = 1$$

u	v	w
0	0	0
1	$\{0,1,\perp\}$	1
$\{0,1,\perp\}$	1	1
Else		$\{0,1,\perp\}$

- If $T =$ AND in $g = (T, u, v, w)$, then $\mathbf{x}$ satisfies

$$\mathbf{x}[u] = \mathbf{x}[v] = 1 \implies \mathbf{x}[w] = 1$$
$$(\mathbf{x}[u] = 0) \vee (\mathbf{x}[v] = 0) \implies \mathbf{x}[w] = 0$$

u	v	w
1	1	1
0	$\{0,1,\perp\}$	0
$\{0,1,\perp\}$	0	0
Else		$\{0,1,\perp\}$

- If $T =$ NAND in $g = (T, u, v, w)$, then $\mathbf{x}$ satisfies

$$\mathbf{x}[u] = \mathbf{x}[v] = 1 \implies \mathbf{x}[w] = 0$$
$$(\mathbf{x}[u] = 0) \lor (\mathbf{x}[v] = 0) \implies \mathbf{x}[w] = 1$$

u	v	w
1	1	0
0	$\{0, 1, \bot\}$	1
$\{0, 1, \bot\}$	0	1
Else		$\{0, 1, \bot\}$

Corollary 2.2. *The* Pure-Circuit *problem is* PPAD*-complete, for any of the following choices of gate types:*

- PURIFY *and at least one of* $\{\mathsf{NOR}, \mathsf{NAND}\}$*;*
- PURIFY*,* NOT*, and at least one of* $\{\mathsf{OR}, \mathsf{AND}\}$*.*

Proof. This follows from Theorem 2.1 by observing that a NOR gate can always be simulated with the given set of gates. Clearly, NOR can be simulated by first using an OR gate and then a NOT gate. Furthermore, OR can be simulated by NOT and AND by applying De Morgan's laws. Finally, AND can easily be obtained from NOT and NAND, and NOT can be obtained from NAND and PURIFY as follows: first apply a PURIFY gate, and then use its two outputs as the two inputs to a NAND gate. □

More structure. The hardness result is also robust with respect to restrictions applied to the *interaction graph*. This graph is constructed on the vertex set $V = [n]$ by adding a directed edge from node u to node v whenever v is the output of a gate with input u. For example, a NOR gate with inputs u, v and output w yields the two edges (u, w) and (v, w). On the other hand, a PURIFY gate with input u and outputs v, w gives the edges (u, v) and (u, w). Since any given node is the output of at most one gate, it immediately follows that the in-degree of every node is at most 2. However, the out-degree of a node can *a priori* be arbitrarily large. It is quite easy to show that the problem remains PPAD-complete, even if we severely restrict the interaction graph.

Corollary 2.3. *The* Pure-Circuit *problem remains* PPAD*-complete, for any choice of gates* $\{\mathsf{PURIFY}, \mathsf{X}, \mathsf{Y}\}$*, where* $(\mathsf{X}, \mathsf{Y}) \in \{\mathsf{NOT}\} \times \{\mathsf{OR}, \mathsf{AND}, \mathsf{NOR}, \mathsf{NAND}\}$ *or* $(\mathsf{X}, \mathsf{Y}) \in \{\mathsf{COPY}\} \times \{\mathsf{NOR}, \mathsf{NAND}\}$ *and even if we also* simultaneously *have all of the following restrictions.*

1. *Every node is the input of exactly one gate.*
2. *In the interaction graph, the total degree of every node is at most 3. More specifically, for every node, the in- and out-degrees,* d_{in} *and* d_{out}*, satisfy* $(d_{in}, d_{out}) \in \{(1, 1), (2, 1), (1, 2)\}$*.*
3. *The interaction graph is bipartite.*

The proof of this corollary is again quite simple, but a bit tedious, so it is relegated to Appendix B.

Remark 1. Using Corollary 2.3, it is also possible to show that Pure-Circuit with only two gates (namely, PURIFY and one of $\{\mathsf{NOR}, \mathsf{NAND}\}$) remains PPAD-complete even if the total degree of every node is at most 4 in the interaction graph. Indeed, NOT gates can be implemented by first using a PURIFY gate and then a NOR/NAND gate. The structural properties of Corollary 2.3 ensure that this yields an interaction graph where the total degree is at most 4 for each node. This can be further reduced to degree 3, if one modifies the definition of Pure-Circuit (Definition 1) so that the two inputs to a NOR/NAND gate are no longer required to be two *distinct* nodes u and v, but can possibly be the same node $u = v$. However, if the definition is modified in that way, then one must be careful when reducing from Pure-Circuit to make sure to take into account the possibility that $u = v$ when constructing the gadget for a NOR/NAND gate.

3 PPAD-completeness of Pure-Circuit

This section proves our main technical result, namely that Pure-Circuit is PPAD-complete (Theorem 2.1). We note that membership in PPAD follows immediately from the reduction of the problem to GCircuit in Section 4.1. In order to establish the PPAD-hardness, we present a polynomial-time reduction from a PPAD-complete problem to Pure-Circuit. The canonical PPAD-complete problem is the End-of-Line problem, but, as is usually the case, we do not reduce directly from End-of-Line, but from a problem with topological structure instead, which we introduce next.

3.1 The StrongSperner Problem

We will reduce from the StrongSperner problem, which is based on a variant of Sperner's lemma [Spe28]. This problem is in essence the same as the HighD-BiSperner problem defined by Daskalakis et al. [DSZ21] and used to prove PPAD-hardness of a problem related to constrained min-max optimization. Furthermore, the corresponding "strong" variant of Tucker's lemma was used by Deligkas et al. [DFHM22] to provide improved PPA-hardness results for the consensus-halving problem in fair division.

Definition 2. The StrongSperner problem:

Input: A Boolean circuit computing a labeling $\lambda : [M]^N \to \{-1,+1\}^N$ satisfying the following boundary conditions for every $i \in [N]$:

- if $x_i = 1$, then $[\lambda(x)]_i = +1$;
- if $x_i = M$, then $[\lambda(x)]_i = -1$.

Output: Points $x^{(1)}, \dots, x^{(N)} \in [M]^N$ that satisfy $\|x^{(i)} - x^{(j)}\|_\infty \leq 1$ for all $i, j \in [N]$, and such that $\lambda(x^{(1)}), \dots, \lambda(x^{(N)})$ *cover all labels*, i.e., for all $i \in [N]$ and $\ell \in \{-1,+1\}$ there exists $j \in [N]$ with $[\lambda(x^{(j)})]_i = \ell$.

Note that the requirement that a solution should consist of exactly N points is without loss of generality. If we find less than N points that cover all labels, then we can simply re-use the same points multiple times to obtain a list of N points that cover all labels (there is no requirement on them being distinct). If we find more than N points that cover all labels, then it is easy to see that we can extract a subset of N points that still cover all labels in polynomial time [DFHM22, Lemma 3.1].

Theorem 3.1 ([DSZ21]). StrongSperner *is* PPAD*-hard, even when M is only polynomially large (i.e., given in unary in the input).*

Remark 2. This was proven by Daskalakis et al. [DSZ21] by reducing from the Succinct-Brouwer problem, which had been proven PPAD-hard by Rubinstein [Rub16]. The PPAD-hardness can also be proved by a more direct reduction from End-of-Line. Indeed, End-of-Line can be reduced to StrongSperner with $N = 2$ and exponentially large M by using the techniques of Chen and Deng [CD09]. Then, a snake embedding technique [CDT09, DFHM22] can be used to obtain hardness for the high-dimensional version with small M, in fact, even for constant M. For our purposes, the hardness for polynomially large M is sufficient.

3.2 Reduction from StrongSperner to Pure-Circuit

Consider an instance $\lambda : [M]^N \to \{-1,+1\}^N$ of StrongSperner, where λ is given as a Boolean circuit and M is only polynomially large (i.e., given in unary). We will now show how to construct an instance of Pure-Circuit in polynomial time such that from any correct assignment to the nodes, we can extract a solution to the StrongSperner instance in polynomial time. We will make use of the gates PURIFY, AND, OR, NOT, COPY. All these gates can easily be simulated using the two gates PURIFY and NOR, by the arguments in the proof of Corollary 2.2.

We begin the construction of the PURE-CIRCUIT instance by creating nodes $u_{i,1}, \dots, u_{i,M}$ for each $i \in [N]$. We call these nodes the *original inputs*, and we think of $u_{i,1}, \dots, u_{i,M}$ as being the unary representation of an element in $[M]$. Of course, this only makes sense when all these nodes are assigned pure bit values, i.e., 0 or 1. In general, this will not be the case. The rest of the instance can be divided into four parts: the *purification* stage, the *circuit* stage, the *sorting* stage, and the *selection* stage.

We begin with a brief overview of the purpose of each stage and how they interact with each other. The purification stage uses the PURIFY gate to create multiple "copies" of the original $u_{i,j}$ nodes, while ensuring that most of the copies have pure bit values (Purification Lemma, Lemma 3.2). Then, the circuit stage evaluates the circuit λ on these copies of the original inputs. Since the purification stage ensures that most copies have pure bits, the circuit stage outputs the correct labels for most copies, and, in particular, most outputs are pure bits (Circuit Lemma, Lemma 3.3). Next, for each $i \in [N]$, the sorting stage "sorts" the list of all ith output values computed in the circuit stage (Sorting Lemma, Lemma 3.4). Since most of the ith output values are pure bits, the sorting stage ensures that all non-pure values are close to each other in the sorted list. The selection stage then proceeds to select M values from the sorted list, but in a careful way, namely such that they are all far away from each other. This ensures that at most one of the M selected values is not a pure bit. The M selected values are then fed back into the original inputs $u_{i,1}, \dots, u_{i,M}$. As a result, the original input $u_{i,1}, \dots, u_{i,M}$ contains at most one non-pure bit, and thus the purification stage ensures that all the (pure) copies of $u_{i,1}, \dots, u_{i,M}$ correspond to unary numbers that differ by at most 1. This means that these copies represent points in the STRONGSPERNER domain that are within ℓ_∞-distance 1 (Selection Lemma, Lemma 3.5). Finally, using the boundary conditions of the STRONGSPERNER instance, we argue that these points must cover all the labels (Solution Lemma, Lemma 3.6).

We now describe each of the stages in more detail. We let K denote the number of copies that we make. It will be enough to pick $K = 3NM^2$. In what follows, $\mathbf{x}$ always denotes an arbitrary solution to the PURE-CIRCUIT instance we construct.

Step 1: Purification stage. For each $(i, j) \in [N] \times [M]$, we construct a binary tree of PURIFY gates that is rooted at $u_{i,j}$ and has leaves $u_{i,j}^{(1)}, \dots, u_{i,j}^{(K)}$.

We say that $k \in [K]$ is a good copy, if $\mathbf{x}[u_{i,j}^{(k)}]$ is a pure bit for all $(i, j) \in [N] \times [M]$. We denote the set of all good copies by G, i.e.,

$$G := \left\{ k \in [K] : \mathbf{x}[u_{i,j}^{(k)}] \in \{0, 1\} \quad \forall (i, j) \in [N] \times [M] \right\}.$$

For a good copy $k \in G$ and any $i \in [N]$, we can interpret the bitstring $(\mathbf{x}[u_{i,j}^{(k)}])_{j \in [M]} \in \{0, 1\}^M$ as representing a number in $[M]$ in unary, which we denote by $u_i^{(k)} \in [M]$. In other words, $u_i^{(k)}$ corresponds to the number of 1's in the bit-string $(\mathbf{x}[u_{i,j}^{(k)}])_{j \in [M]}$. For notational convenience we let the all-zero bit string 0^M correspond to 1 as well, i.e., if $\mathbf{x}[u_{i,j}^{(k)}] = 0$ for all $j \in [M]$, then $u_i^{(k)} = 1$. (Alternatively, we could also use $M - 1$ bits instead of M.) Finally, we let $u^{(k)} \in [M]^N$ denote the vector $(u_1^{(k)}, \dots, u_N^{(k)})$.

Lemma 3.2 (Purification Lemma)**.** *The following hold.*

1. *There are at least $K - NM$ good copies, i.e., $|G| \geq K - NM$.*
2. *If for some $(i, j) \in [N] \times [M]$ the original input $\mathbf{x}[u_{i,j}]$ is a pure bit, then all copies have that same bit, i.e., $\mathbf{x}[u_{i,j}^{(k)}] = \mathbf{x}[u_{i,j}]$ for all $k \in [K]$.*

Proof. Observe that in a binary tree of PURIFY gates, if some node has some pure value $b \in \{0,1\}$, then all nodes in the subtree rooted at this node also have value b. Applying this observation at the root of the tree rooted at $u_{i,j}$, we immediately obtain part 2 of the statement.

For part 1, note that for any $(i,j) \in [N] \times [M]$, in the binary tree of PURIFY gates rooted at $u_{i,j}$, all leaves, except at most one, have a pure bit value. This follows from the definition of the PURIFY gate and the observation about subtrees made in the previous paragraph. As a result, all values $(\mathbf{x}[u_{i,j}^{(k)}])_{(i,j,k)\in[N]\times[M]\times[K]}$ are pure bits, except for at most NM of them. But this means that there are at least $K - NM$ values of $k \in [K]$ such that $\mathbf{x}[u_{i,j}^{(k)}] \in \{0,1\}$ for all $\forall (i,j) \in [N] \times [M]$. In other words, $|G| \geq K - NM$. $\square$

Step 2: Circuit stage. We assume, without loss of generality, that λ is given as a Boolean circuit $C : (\{0,1\}^M)^N \to \{0,1\}^N$ using gates AND, OR, NOT, and, on input $z \in (\{0,1\}^M)^N$:

- For each $i \in [N]$, the ith block of input bits $z_i \in \{0,1\}^M$ is interpreted by C as representing a number $\overline{z_i} \in [M]$ in unary, in the exact same way as $u_i^{(k)} \in [M]$ is obtained from the bitstring $(\mathbf{x}[u_{i,j}^{(k)}])_{j\in[M]}$.
- The circuit C outputs $\lambda(\overline{z_1}, \dots, \overline{z_N}) \in \{-1,+1\}^N$, where a -1 output is represented by a 0, and a $+1$ output by a 1. To keep things simple, in the rest of this exposition we will abuse notation and think of λ as outputting labels in $\{0,1\}^N$.

If the circuit is not originally in this form, then it can be brought in this form in polynomial time.

In the circuit stage, we construct K separate copies of the circuit C, using the AND, OR, and NOT gates. For each $k \in [K]$, the kth copy C_k takes as input the nodes $(u_{i,j}^{(k)})_{(i,j)\in[N]\times[M]}$ and we denote its output nodes by $v_1^{(k)}, \dots, v_N^{(k)}$. Since the gates always have correct output when the inputs are pure bits, we immediately obtain the following.

Lemma 3.3 (Circuit Lemma)**.** *For all good copies $k \in G$, the output of circuit C_k is correct, i.e., $\mathbf{x}[v_i^{(k)}] = [\lambda(u^{(k)})]_i$ for all $i \in [N]$.*

Step 3: Sorting stage. In this stage, for each $i \in [N]$, we would like to have a gadget that takes as input the list of nodes $v_i^{(1)}, \dots, v_i^{(K)}$ (namely, the list of ith outputs of the circuits $C_1, \dots, C_K$) and outputs the nodes $w_i^{(1)}, \dots, w_i^{(K)}$, such that these output nodes are a sorted list of the values of the input nodes (where we think of the values as being ordered $0 < \bot < 1$). Unfortunately, this is not possible given the gates we have at our disposal. However, it turns out that we can do some kind of "weak" sorting by using the robustness of the AND and OR gates (i.e., the fact that AND on input 0 and s, always outputs 0, no matter what $s \in \{0,1,\bot\}$ is).

For now assume that we consider values in $[0,1]$ (instead of $\{0,1,\bot\}$) and that we have access to a comparator gate that takes two inputs s_1 and s_2 and outputs t_1 and t_2, such that t_1, t_2 is the sorted list s_1, s_2. Formally, we can write this as $t_1 := \min\{s_1, s_2\}$ and $t_2 := \max\{s_1, s_2\}$. Using comparator gates, it is easy to construct a circuit that takes K inputs and outputs them in sorted order. Indeed, we can directly implement a sorting network [Knu98], for example. Even a very naive approach will yield such a circuit of polynomial size, which is all we need. We implement this circuit with inputs $v_i^{(1)}, \dots, v_i^{(K)}$ and outputs $w_i^{(1)}, \dots, w_i^{(K)}$ in our Pure-Circuit instance, by replacing every comparator gate by AND and OR gates. Namely, to implement a comparator gate with inputs s_1, s_2 and outputs t_1, t_2, we use an AND gate with inputs s_1, s_2 and output t_1, and an OR gate with inputs s_1, s_2 and output t_2. The robustness of the AND and OR gates allows us to prove that this sorting gadget sorts the pure bit values correctly, in the following sense.

Lemma 3.4 (Sorting Lemma). *Let K_0 and K_1 denote the number of zeroes and ones that the ith sorting gadget gets as input, i.e., $K_b := |\{k \in [K] : \mathbf{x}[v_i^{(k)}] = b\}|$. Then, the first K_0 outputs of the gadget are zeroes, and the last K_1 outputs are ones. Formally, $\mathbf{x}[w_i^{(k)}] = 0$ for all $k \in [K_0]$, and $\mathbf{x}[w_i^{(K+1-k)}] = 1$ for all $k \in [K_1]$.*

Proof. Consider the ideal sorting circuit (that uses comparator gates) with input $f(\mathbf{x}[v_i^{(1)}]), \dots, f(\mathbf{x}[v_i^{(K)}])$, where f maps 0 to 0, 1 to 1, and $\perp$ to 1/2. In other words, we imagine running the comparator circuit on our list of values, except that the "garbage" value $\perp$ is replaced by 1/2. Since the comparator circuit correctly sorts the list, its output satisfies the desired property: the first K_0 outputs are 0, and the last K_1 outputs are 1. Thus, in order to prove the lemma, it suffices to prove the following claim: *if a node in the ideal circuit has a pure bit value $b \in \{0, 1\}$, then the corresponding node in our* Pure-Circuit *instance must also have value b.*

We prove the claim by induction. Clearly, all input nodes satisfy the claim. Now consider some node t_1 that is the min-output of a comparator gate with inputs s_1 and s_2, that both satisfy the claim. Recall that this gate will be implemented in the Pure-Circuit by an AND gate with inputs s_1, s_2 and output t_1. If the ideal circuit assigns value 1/2 to t_1, then the claim trivially holds for t_1. If the ideal circuit assigns value 1 to t_1, then both s_1 and s_2 must have value 1 in the ideal circuit. Since the claim holds for s_1 and s_2, they also have value 1 in Pure-Circuit, and so the AND gate will ensure that t_1 also has value 1, thus satisfying the claim. Finally, if the ideal circuit assigns value 0 to t_1, then it must also have assigned value 0 to at least one of s_1 or s_2. But then, by the claim, Pure-Circuit also assigns value 0 to at least one of s_1 or s_2, and the robustness of the AND gate ensures that t_1 also has value 0. The same argument also works with max and OR instead. □

Note that Lemma 3.4 only guarantees a "weak" type of sorting: some parts of the output list might not be correctly ordered, and the list of output values might not be a permutation of the input values (namely, it can happen that there are more 0's and/or 1's in the output list than in the input list). However, this "weak" sorting will be enough for our needs as we will see below.

Step 4: Selection stage. Since the list $w_i^{(1)}, \dots, w_i^{(K)}$, is now "sorted", we can select M nodes from it in such a way that at most one node does not have a pure value. Indeed, this can be achieved by selecting nodes that are sufficiently far apart from each other. We thus select the nodes $(w_i^{(j \cdot 2NM)})_{j \in [M]}$ and copy their values onto the original input nodes $(u_{i,j})_{j \in [M]}$. Namely, for each $j \in [M]$ we introduce a COPY gate with input $w_i^{(j \cdot 2NM)}$ and output $u_{i,j}$. Recall that $K = 3NM^2 \geq M \cdot 2NM$, so this is well defined. This selection procedure has the following nice properties.

Lemma 3.5 (Selection Lemma). *We have:*

1. *All good copies are close to each other: $\|u^{(k)} - u^{(k')}\|_\infty \leq 1$ for all $k, k' \in G$.*

2. *If all good copies agree that the ith output is $b \in \{0, 1\}$, i.e., $\mathbf{x}[v_i^{(k)}] = b$ for all $k \in G$, then the original input satisfies $\mathbf{x}[u_{i,j}] = b$ for all $j \in [M]$.*

Proof. We begin by proving part 1 of the statement. First of all, note that it suffices to show that, for all $i \in [N]$, at most one of the original inputs $\mathbf{x}[u_{i,1}], \dots, \mathbf{x}[u_{i,M}]$ is not a pure bit. Indeed, in that case, by part 2 of the Purification Lemma (Lemma 3.2), it follows that for any $i \in [N]$ and any $k, k' \in G$, the bitstrings $(\mathbf{x}[u_{i,j}^{(k)}])_{j \in [M]}$ and $(\mathbf{x}[u_{i,j}^{(k')}])_{j \in [M]}$ differ in at most one bit. But this implies that $|u_i^{(k)} - u_i^{(k')}| \leq 1$ for all $i \in [N]$, and thus $\|u^{(k)} - u^{(k')}\|_\infty \leq 1$.

Now consider any $i \in [N]$. By part 1 of the Purification Lemma (Lemma 3.2), we have that the number of good copies $|G| \geq K - NM$. By the Circuit Lemma (Lemma 3.3) we know that the corresponding outputs are pure bits, i.e., $\mathbf{x}[v_i^{(k)}] \in \{0,1\}$ for all $k \in G$. Thus, the list $\mathbf{x}[v_i^{(1)}], \ldots, \mathbf{x}[v_i^{(K)}]$ contains at least $K - NM$ pure bits. Using the notation from the Sorting Lemma (Lemma 3.4), this means that $K_0 + K_1 \geq K - NM$. As a result, by applying the Sorting Lemma, it follows that in the list obtained after sorting, $\mathbf{x}[w_i^{(1)}], \ldots, \mathbf{x}[w_i^{(K)}]$, all the non-pure bits are contained in an interval of length NM. Since the selected nodes $(w_i^{(j \cdot 2NM)})_{j \in [M]}$ are sufficiently far apart (namely $2NM$), at most one such node can fall in the "bad" interval. This means that all selected nodes, except at most one, are pure bits, and since the selected nodes are copied into the original inputs, this also holds for them.

It remains to prove part 2 of the statement. If for some $i \in [N]$, $\mathbf{x}[v_i^{(k)}] = 0$ for all $k \in G$, then this means that $K_0 \geq |G| \geq K - NM$. By the Sorting Lemma (Lemma 3.4), it follows that $\mathbf{x}[w_i^{(k)}] = 0$ for all $k \in [K - NM]$. In particular, since $K - NM = 3NM^2 - NM \geq M \cdot 2NM$, this means that for all $j \in [M]$, $\mathbf{x}[w_i^{(j \cdot 2NM)}] = 0$. But these are the nodes we select, and we copy their value into the original inputs, so the statement follows. The case where $\mathbf{x}[v_i^{(k)}] = 1$ for all $k \in G$ is handled analogously. □

Correctness. The description of the reduction is now complete. We have constructed a valid instance of Pure-Circuit in polynomial time. In particular, note that every node is the output of exactly one gate. To complete the proof, it remains to prove that from any solution of the Pure-Circuit instance we can extract a solution to StrongSperner in polynomial time. We do this in the following final lemma.

Lemma 3.6 (Solution Lemma). *The points $\{u^{(k)} : k \in G\} \subseteq [M]^N$ yield a solution to the* StrongSperner *instance λ.*

Proof. First of all, by part 1 of the Selection Lemma (Lemma 3.5), we know that the points $u^{(k)}$, $k \in G$, are all within ℓ_∞-distance 1. Thus, it suffices to prove that they cover all labels with respect to λ. Note that we can efficiently extract these points from a solution $\mathbf{x}$, since, for each $k \in [K]$, we can easily decide whether k lies in G or not, and then extract the point $u^{(k)}$.

We prove by contradiction that the points $u^{(k)}$, $k \in G$, must cover all labels. Assume, on the contrary, that there exists $i \in [N]$ and $b \in \{0,1\}$ such that $[\lambda(u^{(k)})]_i = b$ for all $k \in G$. Then, by the Circuit Lemma (Lemma 3.3), all the corresponding circuits must output b, i.e., $\mathbf{x}[v_i^{(k)}] = b$ for all $k \in G$. By part 2 of the Selection Lemma (Lemma 3.5), it follows that the original input satisfies $\mathbf{x}[u_{i,j}] = b$ for all $j \in [M]$. Finally, by part 2 of the Purification Lemma (Lemma 3.2), it must be that for all copies $k \in G$, all M bits $\mathbf{x}[u_{i,1}^{(k)}], \ldots, \mathbf{x}[u_{i,M}^{(k)}]$ are equal to b. Now, if $b = 1$, then this means that $u_i^{(k)} = M$, and thus $[\lambda(u^{(k)})]_i = 0 \neq b$ by the StrongSperner boundary conditions, which contradicts the original assumption. Similarly, if $b = 0$, then $u_i^{(k)} = 1$, and thus $[\lambda(u^{(k)})]_i = 1 \neq b$ by the StrongSperner boundary conditions, which is again a contradiction. (We recall here that we have renamed the labels $\{-1,+1\}$ to $\{0,1\}$, respectively, for the purpose of this proof.) □

4 Applications

In this section, we derive strong inapproximability lower bounds for PPAD-complete problems, by reducing from Pure-Circuit.

4.1 Generalized Circuit

The ε-GCircuit problem was introduced by Chen, Deng, and Teng [CDT09]. In this section, we show that ε-GCircuit is PPAD-hard for all $\varepsilon < 0.1$.

The ε-GCircuit problem. The problem is defined as follows.

Definition 3 (Generalized Circuit [CDT09])**.** A generalized circuit is a tuple (V, T), where V is a set of nodes, and T is a set of gates. Each gate $t \in T$ is a five-tuple (G, u, v, w, c), where G is a gate type from the set $\{G_c, G_{\times c}, G_=, G_+, G_-, G_<, G_\vee, G_\wedge, G_\neg\}$, $u, v \in V \cup \{\mathsf{nil}\}$ are input variables, $w \in V$ is an output variable, and $c \in [0,1] \cup \{\mathsf{nil}\}$ is a rational constant.

The following requirements must be satisfied for each gate $(G, u, v, w, c) \in T$.

- G_c gates take no input variables and use a constant in $[0,1]$. So $u = v = \mathsf{nil}$ and $c \in [0,1]$ whenever $G = G_c$.
- $G_{\times c}$ gates take one input variable and a constant. So $u \in V$, $v = \mathsf{nil}$, and $c \in [0,1]$ whenever $G = G_{\times c}$.
- $G_=$ and $G_\neg$ gates take one input variable and do not use a constant. So $u \in V$, $v = c = \mathsf{nil}$, whenever $G \in \{G_=, G_\neg\}$.
- All other gates take two input variables and do not use a constant. So $u \in V$, $v \in V$, and $c = \mathsf{nil}$ whenever $G \notin \{G_c, G_{\times c}, G_=, G_\neg\}$.
- Every variable in V is the output variable for exactly one gate. More formally, for each variable $w \in V$, there is exactly one gate $t \in T$ such that $t = (G, u, v, w, c)$.

The ε-GCircuit problem is defined as follows. Given a generalized circuit (V, T), find a vector $\mathbf{x} \in [0,1]^{|V|}$ such that for each gate in T the following constraints are satisfied.

Gate	Constraint
$(G_c, \mathsf{nil}, \mathsf{nil}, w, c)$	$\mathbf{x}[w] = c \pm \varepsilon$
$(G_{\times c}, u, \mathsf{nil}, w, c)$	$\mathbf{x}[w] = \mathbf{x}[u] \cdot c \pm \varepsilon$
$(G_=, u, \mathsf{nil}, w, \mathsf{nil})$	$\mathbf{x}[w] = \mathbf{x}[u] \pm \varepsilon$
$(G_+, u, v, w, \mathsf{nil})$	$\mathbf{x}[w] = \min(\mathbf{x}[u] + \mathbf{x}[v], 1) \pm \varepsilon$
$(G_-, u, v, w, \mathsf{nil})$	$\mathbf{x}[w] = \max(\mathbf{x}[u] - \mathbf{x}[v], 0) \pm \varepsilon$
$(G_<, u, v, w, \mathsf{nil})$	$\mathbf{x}[w] = \begin{cases} 1 \pm \varepsilon & \text{if } \mathbf{x}[u] < \mathbf{x}[v] - \varepsilon \\ 0 \pm \varepsilon & \text{if } \mathbf{x}[u] > \mathbf{x}[v] + \varepsilon \end{cases}$
$(G_\vee, u, v, w, \mathsf{nil})$	$\mathbf{x}[w] = \begin{cases} 1 \pm \varepsilon & \text{if } \mathbf{x}[u] \geq 1 - \varepsilon \text{ or } \mathbf{x}[v] \geq 1 - \varepsilon \\ 0 \pm \varepsilon & \text{if } \mathbf{x}[u] \leq \varepsilon \text{ and } \mathbf{x}[v] \leq \varepsilon \end{cases}$
$(G_\wedge, u, v, w, \mathsf{nil})$	$\mathbf{x}[w] = \begin{cases} 1 \pm \varepsilon & \text{if } \mathbf{x}[u] \geq 1 - \varepsilon \text{ and } \mathbf{x}[v] \geq 1 - \varepsilon \\ 0 \pm \varepsilon & \text{if } \mathbf{x}[u] \leq \varepsilon \text{ or } \mathbf{x}[v] \leq \varepsilon \end{cases}$
$(G_\neg, u, \mathsf{nil}, w, \mathsf{nil})$	$\mathbf{x}[w] = \begin{cases} 1 \pm \varepsilon & \text{if } \mathbf{x}[u] \leq \varepsilon \\ 0 \pm \varepsilon & \text{if } \mathbf{x}[u] \geq 1 - \varepsilon \end{cases}$

Here the notation $a = b \pm \varepsilon$ is used as a shorthand for $a \in [b - \varepsilon, b + \varepsilon]$. We will also make use of gates of type $(G_>, u, v, w, \mathsf{nil})$ which enforce the constraint

$$\mathbf{x}[w] = \begin{cases} 0 \pm \varepsilon & \text{if } \mathbf{x}[u] < \mathbf{x}[v] - \varepsilon \\ 1 \pm \varepsilon & \text{if } \mathbf{x}[u] > \mathbf{x}[v] + \varepsilon \end{cases}$$

Gates of type $G_>$ can be easily built by using a $G_\neg$ gate to negate the output of a $G_<$ gate.

In the remainder of this section, we prove the following result.

Theorem 4.1. *ε-GCircuit is PPAD-hard for every $\varepsilon < 0.1$.*

Proof. We will reduce from the Pure-Circuit problem that uses the gates NOR and PURIFY, which we showed to be PPAD-hard in Theorem 2.1. We will encode 0 values in the Pure-Circuit problem as values in the range $[0, \varepsilon]$ in GCircuit, while 1 values will be encoded as values in the range $[1 - \varepsilon, 1]$. Then, each gate from Pure-Circuit will be simulated by a combination of gates in the GCircuit instance.

NOR gates. A NOR gate (NOR, u, v, w) will be simulated by GCircuit gates that compute

$$\mathbf{x}[u] + \mathbf{x}[v] < 5/9,$$

which requires us to use G_+, $G_<$ and G_c. We claim that this gate works for any $\varepsilon < 1/9$.

The idea is that if both inputs lie in the range $[0, \varepsilon]$, and thus both inputs encode zeros in Pure-Circuit, we will have $\mathbf{x}[u] + \mathbf{x}[v] \leq 3\varepsilon < 3/9$, where the extra ε error is introduced by G_+. The G_c gate outputting $5/9$ may output a value as small as $4/9$ after the error is taken into account. Thus, the $G_<$ gate will compare these two values, and provided that $\varepsilon < 1/9$, the comparison will succeed, so the gate will output a value greater than or equal to $1 - \varepsilon$, which corresponds to a 1 in the Pure-Circuit instance, as required.

If, on the other hand, at least one input is in the range $[1 - \varepsilon, 1]$, then we will have $\mathbf{x}[u] + \mathbf{x}[v] \geq 1 - 2\varepsilon > 7/9$, where again G_+ introduces an extra ε error. The G_c gate outputting $5/9$ may output a value as large as $6/9$ after the error is taken into account. Thus, the $G_<$ will compare these two values and provided that $\varepsilon < 1/9$, the comparison will succeed, so the gate will output a value less than or equal to ε, which corresponds to a 0 in the Pure-Circuit instance, which is again as required.

PURIFY gates. A PURIFY gate $(\mathsf{PURIFY}, u, v, w)$ will be simulated by two GCircuit gadgets. The first will set $\mathbf{x}[v]$ equal to $\mathbf{x}[u] > 0.3$, while the second will set $\mathbf{x}[w]$ equal to $\mathbf{x}[u] > 0.7$, where G_c and $G_>$ gates are used to implement these operations. We claim that this construction works for any $\varepsilon < 0.1$.

We begin by considering the first gadget, which sets $\mathbf{x}[v]$ equal to $\mathbf{x}[u] > 0.3$. If $\mathbf{x}[u] \leq \varepsilon < 0.1$, then note that the G_c gate outputting 0.3 may output a value as small as 0.2 once errors are taken into account. Thus, since $\varepsilon < 0.1$, the comparison made by the $G_>$ gate will succeed, and so $\mathbf{x}[v]$ will be set to a value less than or equal to ε. On the other hand, if $\mathbf{x}[u] \geq 0.5$, then note that the G_c gate outputting 0.3 may output a value as large as 0.4 once errors are taken into account. So, the comparison made by the $G_>$ gate will again succeed, and $\mathbf{x}[v]$ will be set to a value greater than or equal to $1 - \varepsilon$.

One can repeat the analysis above for the second gadget to conclude that $\mathbf{x}[w]$ will be set to a value less than or equal to ε when $\mathbf{x}[u] \leq 0.5$, and a value greater than or equal to $1 - \varepsilon$ when $\mathbf{x}[u] \geq 1 - \varepsilon$. So we can verify that the conditions of the PURIFY gate are correctly simulated.

- If $\mathbf{x}[u] \leq \varepsilon$, meaning that the input encodes a zero, then $\mathbf{x}[v]$ and $\mathbf{x}[w]$ will be set to values that are less than or equal to ε, and so both outputs encode zeros.
- If $\mathbf{x}[u] \geq 1 - \varepsilon$, meaning that the input encodes a one, then $\mathbf{x}[v]$ and $\mathbf{x}[w]$ will be set to values that are greater than or equal to $1 - \varepsilon$, and so both outputs encode ones.
- No matter what value $\mathbf{x}[u]$ takes, at least one output will be set to a value that encodes a zero or a one. Specifically, if $\mathbf{x}[u] \leq 0.5$, then the second comparison gate will set $\mathbf{x}[w] \leq \varepsilon$, while if $\mathbf{x}[u] \geq 0.5$, then the first comparison gate will set $\mathbf{x}[v] \geq 1 - \varepsilon$.

Thus, the construction correctly simulates a PURIFY gate. Note that ε cannot be increased beyond 0.1 in this construction, since then there would be no guarantee that an encoding of a zero or a one would be produced when $\mathbf{x}[u] = 0.5$.

The lower bound. Given a Pure-Circuit instance defined over variables V, we produce a GCircuit instance by replacing each gate in the Pure-Circuit with the constructions given above. Then, given a solution $\mathbf{x}$ to the 0.1-GCircuit instance, we can produce a solution $\mathbf{x'}$ to Pure-Circuit in the following way. For each $v \in V$

- if $\mathbf{x}[v] \leq \varepsilon$, then we set $\mathbf{x'}[v] = 0$,

- if $\mathbf{x}[v] \geq 1 - \varepsilon$, then we set $\mathbf{x'}[v] = 1$, and
- if $\mathbf{x}[v] > \varepsilon$ and $\mathbf{x}[v] < 1 - \varepsilon$, then we set $\mathbf{x'}[v] = \perp$.

By the arguments given above, we have that $\mathbf{x'}$ is indeed a solution to PURE-CIRCUIT, and thus Theorem 4.1 is proved. $\square$

4.2 Polymatrix Games

In this section, we show a number of results for polymatrix games. After defining this class of games and the equilibrium notions of interest, we provide a simple algorithm that computes a 1/3-well-supported Nash equilibrium (WSNE) in polynomial time (Theorem 4.2). We then show that this algorithm is in fact optimal, by proving hardness of finding ε-WSNE for all $\varepsilon < 1/3$ (Theorem 4.3). Next, we also prove hardness of computing ε-Nash equilibria for all $\varepsilon < 1/48$ (Theorem 4.4). Both of these hardness results hold for degree 3 bipartite games with at most two strategies per player. Finally, we show that the hardness result for ε-WSNE also holds for win-lose polymatrix games in degree 7 bipartite games with at most two strategies per player (Theorem 4.5).

Definitions. A *polymatrix game* is defined by an undirected graph (V, E), where each vertex represents a player, and we use $n = |V|$ to denote the number of players in the game. Each player $i \in V$ has a fixed number of actions (also called pure strategies) given by m_i. For each edge $(i, j) \in E$, there is an $m_i \times m_j$ matrix A_{ij} giving the payoffs that player i obtains from their interaction with player j, and likewise there is an $m_j \times m_i$ matrix A_{ji} giving payoffs for player j's interaction with player i.

A *mixed strategy* for player i specifies a probability distribution over player i's actions, so if $\Delta^k := \{x \in \mathbb{R}^{k+1} : x \geq 0, \sum_{i=1}^{k+1} x_i = 1\}$ denotes the k-dimensional simplex, we have that Δ^{m_i-1} is the set of mixed strategies for player i. The *support* of a mixed strategy $s_i = (s_i(1), s_i(2), \ldots, s_i(m_i)) \in \Delta^{m_i-1}$ is given by $\mathsf{supp}(s_i) = \{j \in \{1, 2, \ldots, m_i\} : s_i(j) > 0\}$, which is the set of pure strategies that are played with non-zero probability in strategy s_i.

A *strategy profile* specifies a mixed strategy for each of the players, and so the set of strategy profiles is given by $\Sigma = \Delta^{m_1-1} \times \cdots \times \Delta^{m_n-1}$. For each strategy profile $\mathbf{s} = (s_1, s_2, \ldots, s_n) \in \Sigma$, the payoff to player i is

$$u_i(\mathbf{s}) := s_i^T \cdot \sum_{j\,:\,(i,j)\in E} A_{ij} \cdot s_j.$$

In other words, the payoff obtained by player i is the sum of the payoffs obtained from the interaction of i with every neighbouring player j. We denote by $\mathbf{s}_{-i}$ (resp. $\mathbf{a}_{-i}$) the *partial strategy profile* (resp. *partial action profile*) consisting of the strategies (resp. actions) of all players except i. For any particular pure strategy $k \in [m_i]$, the payoff when player i plays k, and all other players play according to $\mathbf{s}$ is denoted by

$$u_i(k, \mathbf{s}_{-i}) := e_k^T \cdot \sum_{j\,:\,(i,j)\in E} A_{ij} \cdot s_j.$$

where e_k is the m_i-dimensional vector with all entries set to 0, except for the k-th that is set to 1.

A pure strategy k is a *best response* for player i in strategy profile $\mathbf{s}$ if it achieves the maximum payoff over all the pure strategies available to player i, meaning that

$$u_i(k, \mathbf{s}_{-i}) = \max_{1\leq \ell \leq m_i} u_i(\ell, \mathbf{s}_{-i}).$$

Strategy k is an ε-*best response* if it is within ε of being a best response, meaning that

$$u_i(k, \mathbf{s}_{-i}) \geq \max_{1\leq \ell \leq m_i} u_i(\ell, \mathbf{s}_{-i}) - \varepsilon.$$

The *best response payoff* for player i is the payoff associated with the best response strategies, which we denote as

$$\mathrm{br}_i(\mathbf{s}_{-i}) := \max_{1 \le \ell \le m_i} u_i(\ell, \mathbf{s}_{-i}).$$

A strategy profile $\mathbf{s}$ is a *Nash equilibrium* if every player achieves their best response payoff, meaning that $\mathrm{br}_i(\mathbf{s}_{-i}) = u_i(\mathbf{s})$ for all players i. The approximation notions that will be of interest to us in this section are the following.

Definition 4 (ε-NE)**.** A strategy profile $\mathbf{s}$ is an *ε-Nash equilibrium* (ε-NE) if every player's payoff is within ε of their best response payoff. Formally,

$$\forall i \in [n], \quad u_i(\mathbf{s}) \ge \mathrm{br}_i(\mathbf{s}_{-i}) - \varepsilon.$$

Definition 5 (ε-WSNE)**.** A strategy profile $\mathbf{s}$ is an *ε-well-supported Nash equilibrium* (ε-WSNE) if every player only plays strategies that are ε-best responses, meaning that for all i we have that $\mathsf{supp}(s_i)$ contains only ε-best response strategies. Formally,

$$\forall i \in [n], \forall k \in \mathsf{supp}(s_i), \quad u_i(k, \mathbf{s}_{-i}) \ge \mathrm{br}_i(\mathbf{s}_{-i}) - \varepsilon.$$

Payoff normalization. Note that ε-NE and ε-WSNE are both additive notions of approximation, and so values of ε cannot normally be compared across games, since doubling all payoffs in the game would double the value of ε, for example. To deal with this, it is common to normalize the payoffs of the game, which allows us to meaningfully compare values of ε across games.

We adopt the normalization scheme for polymatrix games given by Deligkas et al. [DFSS17], which is a particularly restrictive scheme, but doing so will allow our lower bounds to be compared to the $0.5 + \delta$ upper bound for finding ε-NEs given in that paper.

The idea is to rescale the payoffs so that each player's maximum possible payoff is 1, and their minimum possible payoff is 0. For each player i we compute

$$\begin{aligned} U_i &= \max_{1 \le p \le m_i} \left(\sum_{j \,:\, (i,j) \in E} \max_{1 \le q \le m_j} (A_{ij})_{(p,q)} \right) \\ L_i &= \min_{1 \le p \le m_i} \left(\sum_{j \,:\, (i,j) \in E} \min_{1 \le q \le m_j} (A_{ij})_{(p,q)} \right). \end{aligned}$$

Then, if $d(i)$ denotes the degree of player i in (V, E), we apply the following transformation to each payoff z in each payoff matrix A_{ij}

$$T(z) = \frac{1}{U_i - L_i} \cdot \left(z - \frac{L_i}{d(i)} \right).$$

In our hardness reductions, we will directly build games that are already normalized, meaning that $U_i = 1$ and $L_i = 0$ for all players i.

4.2.1 A Simple Algorithm for 1/3-WSNE in Two-Action Polymatrix Games

In this section we present a simple polynomial-time algorithm for computing a 1/3-WSNE in a two-action polymatrix game. We begin with a description of the algorithm. The algorithm proceeds in two steps:

Step 1: Eliminating players who don't need to mix. In the first step of the algorithm, we check if there exists some player i such that one of its two actions is always a 1/3-best response, no matter what actions the other players pick. More formally, we check if there exists a player i and an action $k \in \{0, 1\}$ such that $u_i(k, \mathbf{s}_{-i}) \geq \max\{u_i(0, \mathbf{s}_{-i}), u_i(1, \mathbf{s}_{-i})\} - 1/3$ for all strategy profiles $\mathbf{s}$. If we find such a player i, then we fix their strategy to always play action k, and we remove them from the game, while also updating the payoffs of the other players to reflect the fact that player i always plays action k. Note that no matter how we fix the strategies of the remaining players later in the algorithm, player i is guaranteed to play a 1/3-best response action in the original game.

After removing player i from the game, and updating the payoffs, we again check if there exists some player j with a 1/3-best response action, and if so, fix their strategy and remove them from the game, as above. After at most n iterations, we are left with a game where none of the remaining players has an action that is always a 1/3-best response.

Step 2: Letting the remaining players mix uniformly. In the second step of the algorithm, we simply fix the strategies of the remaining players so that they mix uniformly between their two actions, i.e., they play action 0 with probability 1/2, and action 1 with probability 1/2.

We note that the simple approach used by the algorithm is essentially the same as an algorithm given by Liu et al. [LLD21] for computation of exact equilibria in very sparse win-lose polymatrix games. We now prove the following.

Theorem 4.2. *The algorithm computes a* 1/3*-WSNE in polynomial time in two-action polymatrix games.*

Proof. The algorithm clearly runs in polynomial time. To prove its correctness, it suffices to show that for the remaining players (i.e., those that were not removed during Step 1) both their actions are 1/3-best-responses, when all the other remaining players mix uniformly.

Consider some player i that was not eliminated in the first step of the algorithm. Define the function $\phi : \Sigma_{-i} \to \mathbb{R}$ by letting $\phi(\mathbf{s}_{-i}) = u_i(0, \mathbf{s}_{-i}) - u_i(1, \mathbf{s}_{-i})$, where $\Sigma_{-i} := (\Delta^1)^{n-1} \equiv [0, 1]^{n-1}$, i.e., $\mathbf{s}_{-i}$ is a partial strategy profile (without the strategy of i). The crucial observation is that ϕ is a linear function, by definition of the utility function in a polymatrix game. Furthermore, by the payoff normalization it holds that $\phi(\mathbf{s}_{-i}) \in [-1, 1]$ for all $\mathbf{s}_{-i} \in \Sigma_{-i}$, since $u_i(0, \mathbf{s}_{-i}), u_i(1, \mathbf{s}_{-i}) \in [0, 1]$.

Since player i was not eliminated in the first step of the algorithm, it must be that none of the two actions 0 and 1 is always a 1/3-best response. In other words, the function ϕ satisfies $\max_{\mathbf{s}_{-i}} \phi(\mathbf{s}_{-i}) > 1/3$ and $\min_{\mathbf{s}_{-i}} \phi(\mathbf{s}_{-i}) < -1/3$. Now, since ϕ is a linear function, the maximum and minimum are achieved at two opposite corners of the domain $\Sigma_{-i} = (\Delta^1)^{n-1} \equiv [0, 1]^{n-1}$. Denote these opposite corners $\ell, h \in \Sigma_{-i}$, such that $\phi(\ell) < -1/3$ and $\phi(h) > 1/3$.

Since ℓ and h are opposite corners of the domain Σ_{-i}, it follows that $\ell/2 + h/2$ corresponds to the partial strategy profile where all players mix uniformly. As a result, it remains to show that $\phi(\ell/2 + h/2) \in [-1/3, 1/3]$, which exactly means that both actions 0 and 1 are 1/3-best responses for player i, when all other players mix uniformly. By linearity of ϕ we can write

$$\phi(\ell/2 + h/2) = \phi(\ell)/2 + \phi(h)/2 \geq -1/2 + (1/3)/2 = -1/3$$

since $\phi(\ell) \geq -1$ and $\phi(h) > 1/3$. The inequality $\phi(\ell/2 + h/2) \leq 1/3$ is similarly obtained by using $\phi(\ell) < -1/3$ and $\phi(h) \leq 1$. We have thus shown that $\phi(\ell/2 + h/2) \in [-1/3, 1/3]$ as desired. □

4.2.2 Hardness for ε-WSNE

In this section we prove that computing an ε-WSNE is PPAD-hard for any $\varepsilon < 1/3$, even in two-action polymatrix games. In particular, this shows that the simple algorithm presented in the previous section is optimal.

Theorem 4.3. *It is* PPAD*-hard to find an* ε*-WSNE in a polymatrix game for all* $\varepsilon < 1/3$*, even in degree 3 bipartite games with two strategies per player.*

Proof. We reduce from the variant of PURE-CIRCUIT that uses NOT, AND, and PURIFY gates, and we will specifically reduce from the hardness result given in Corollary 2.3, since it will give us extra properties for the hard polymatrix games.

Given an instance (V, G) of PURE-CIRCUIT, we will build a polymatrix game with vertex set V, meaning each player in the game will simulate a variable from PURE-CIRCUIT. Each player i will have exactly two strategies, which we will denote as zero and one. The strategy s_i of player i will encode the value of the variable in the following way.

- If s_i places all probability on zero, then this corresponds to setting variable i to 0.
- If s_i places all probability on one, then this corresponds to setting variable i to 1.
- If s_i is a strictly mixed strategy, then this corresponds to setting variable i to $\perp$.

The edges of the game will simulate the gates. The definition of PURE-CIRCUIT requires that each variable v is the output of exactly one gate g. An important property of our reduction is that the player representing v only receives non-zero payoffs from the inputs to the gate g, and receives zero payoffs from any other gates that use v as an input. This means that we can reason about the equilibrium condition of each of the gates independently, without worrying about where the values computed by those gates are used.

NOT gates. For a gate $g = (\mathsf{NOT}, u, v)$, the player v, who represents the output variable, will play the following bimatrix game against u, who represents the input variable.

v \ u	zero	one
zero	0 0	0 1
one	0 1	0 0

This depiction of a bimatrix game shows the matrices A_{vu} and A_{uv}, with the payoffs for v being shown in the bottom-left of each cell, and the payoffs for u being shown in the top-right.

We claim that this gate will work for all $\varepsilon < 1$.

- If u plays zero as a pure strategy, then one gives payoff 1 to v and zero gives payoff 0 to v. So in any ε-WSNE with $\varepsilon < 1$, only one can be an ε-best response for v, meaning that v must play one as a pure strategy, as required by the constraints of the NOT gate.
- Using identical reasoning, if u plays one as a pure strategy, then v must play zero as a pure strategy in any ε-WSNE.
- If u plays a strictly mixed strategy, then the NOT gate places no constraints on the output, so we need not prove anything about v's strategy.

AND gates. For a gate $g = (\mathsf{AND}, u, v, w)$, the player w, who represents the output variable, will play the following matrix games against u and v, who represent the input variables.

w \ u	zero	one
zero	$\frac{1}{2}$, 0	0, 0
one	0, 0	$\frac{1}{6}$, 0

w \ v	zero	one
zero	$\frac{1}{2}$, 0	0, 0
one	0, 0	$\frac{1}{6}$, 0

We claim that this gate will work for all $\varepsilon < \frac{1}{3}$.

- If both u and v play one as a pure strategy, then the payoff to w for playing zero is 0, and the payoff to w for playing one is $1/6 + 1/6 = 1/3$. So, if $\varepsilon < \frac{1}{3}$, then in all ε-WSNEs the only ε-best response for w is one, as required by the constraints of the AND gate.
- If at least one of u and v play zero as a pure strategy, then the payoff to w for playing zero is at least $1/2$, while the payoff to w for playing one is at most $1/6$. So, if $\varepsilon < \frac{1}{3}$, then in all ε-WSNEs the only ε-best response for w is zero, as required by the constraints of the AND gate.
- The AND gate places no other constraints on the variable w, so we can ignore all other cases, e.g., the case where both u and v play strictly mixed strategies.

PURIFY gates. For a gate $g = (\mathsf{PURIFY}, u, v, w)$, the players v and w, who represent the output variables, play the following games against u, who represents the input variable.

v \ u	zero	one
zero	$\frac{1}{3}$, 0	0, 0
one	0, 0	1, 0

w \ u	zero	one
zero	1, 0	0, 0
one	0, 0	$\frac{1}{3}$, 0

We claim that this gate works for any $\varepsilon < 1/3$. We first consider player v.

- If u plays zero as a pure strategy, then strategy zero gives payoff $1/3$ to v, while strategy one gives payoff 0, so in any ε-WSNE with $\varepsilon < 1/3$ we have that zero is the only ε-best response for v.
- If u places at least $1/2$ probability on one, then strategy zero gives at most $1/6$ payoff to v, while strategy one gives at least $1/2$ payoff to v. So in any ε-WSNE with $\varepsilon < 1/3$ we have that one is the only ε-best response for v.

Symmetrically, for player w we have the following.

- If u places at most $1/2$ probability on one, then strategy zero gives at least $1/2$ payoff to w, while strategy one gives at most $1/6$ payoff to w. So in any ε-WSNE with $\varepsilon < 1/3$ we have that zero is the only ε-best response for w.
- If u plays one as a pure strategy, then strategy one gives payoff $1/3$ to w, while strategy zero gives payoff 0, so in any ε-WSNE with $\varepsilon < 1/3$ we have that one is the only ε-best response for w.

From these properties, we can verify that the constraints imposed by the PURIFY gate are enforced correctly.

- If u plays zero as a pure strategy, then both v and w play zero as a pure strategy.
- If u plays one as a pure strategy, then both v and w play one as a pure strategy.
- No matter how u mixes, at least one of v and w plays a pure strategy, with v playing pure strategy zero whenever u places at most 0.5 probability on one, and w playing pure strategy one whenever u places at least 0.5 probability on one.

Hardness result. As we have argued above, each of the gate constraints from the Pure-Circuit instance are enforced correctly in any ε-WSNE of the polymatrix game with $\varepsilon < 1/3$. Thus, given such an ε-WSNE, we can produce a satisfying assignment to the Pure-Circuit instance using the mapping that we defined at the start of the reduction. All of the games that we have presented are already normalized so that each player's maximum possible payoff is 1, and minimum possible payoff is 0, so no extra normalization step is necessary.

Since we are reducing from Corollary 2.3, we get some extra properties about the structure of the polymatrix game. Observe that the interaction graph of the polymatrix game is exactly the interaction graph of the Pure-Circuit instance, as defined in Section 2. Thus Corollary 2.3 implies that the polymatrix game is degree three and bipartite. Moreover, by construction, each player has exactly two strategies. □

4.2.3 Hardness for ε-NE

We now show that computing an ε-NE in polymatrix games is PPAD-complete for all $\varepsilon \leq 1/48$ even in degree 3 bipartite games with two strategies per player. It is worth noting that, given our PPAD-hardness result for ε'-WSNE for all $\varepsilon' < 1/3$ in polymatrix games (Theorem 4.3), one can effortlessly get PPAD-hardness for ε-NE for ε smaller than roughly 0.00136 by Lemma 7.4 of [Rub18]. However, by doing a direct reduction from Pure-Circuit we get an upper bound for ε that is more than 15 times greater.

Theorem 4.4. *It is PPAD-hard to find an ε-NE in a polymatrix game for all $\varepsilon \leq 1/48$, even in degree 3 bipartite games with two strategies per player.*

Proof. The construction itself is identical to the one given in the previous section for ε-WSNE. That is, we build an identical game, using identical gadgets, but for ε-NE we give a different analysis. Whereas for ε-WSNE, we used pure strategies to represent 0 and 1 values in Pure-Circuit, for ε-NE we give a looser mapping that also allows mixed strategies to encode 0s and 1s. Specifically, given a *cutoff* $\delta \in (0, 1/2)$ we map the strategies of the players to values in the Pure-Circuit in the following way.

- If s_i places probability $[1 - \delta, 1]$ on zero, then this corresponds to setting variable i to 0.
- If s_i places probability $[1 - \delta, 1]$ on one, then this corresponds to setting variable i to 1.
- Otherwise, this corresponds to setting variable i to $\perp$.

We show that if $\delta = 1/8$ and $\varepsilon \leq 1/48$, then the gadgets for NOT, AND, and PURIFY continue to work, and any ε-NE of the game corresponds to a solution for Pure-Circuit.

NOT gates. For a gate $g = (\mathsf{NOT}, u, v)$, the player v, who represents the output variable, will play the same bimatrix game against u as that in the ε-WSNE case (see figure in Section 4.2.2).

We claim that this gadget will correctly simulate the NOT gate for $\delta = 1/8$ and all $\varepsilon \le 3/32$ (and therefore all $\varepsilon \le 1/48$). Let u's strategy be $(1-q, q)$ and v's strategy be $(1-p, p)$, where $q, p \in [0, 1]$, for playing zero and one, respectively. We need to show that in any ε-NE,

(i) if $q \le \delta$ then $p \ge 1 - \delta$,

(ii) if $q \ge 1 - \delta$ then $p \le \delta$.

The expected payoff of v is $U_v(p) = (1-p)q + p(1-q) = p + q(1-2p)$ while her expected payoff for playing pure strategy zero is $U_v(0) = q$ and for pure strategy one is $U_v(1) = 1-q$. Then in any ε-NE, in case (i) we have $U_v(p) \ge U_v(1) - \varepsilon$, or equivalently, $(1-p)(1-2q) \le \varepsilon$. Suppose $p < 1-\delta$. Since $q \le \delta$ and $1-2\delta > 0$, we have $(1-p)(1-2q) \ge (1-p)(1-2\delta) > \delta(1-2\delta) = 3/32$. Therefore, for any $\varepsilon \le 3/32$ this is not an ε-NE, and so we conclude that $p \ge 1 - \delta$, as required.

Similarly, for case (ii), in any ε-NE, we have $U_v(p) \ge U_v(0) - \varepsilon$, or equivalently, $p(2q-1) \le \varepsilon$. Suppose $p > \delta$. Since $q \ge 1 - \delta$ and $2(1-\delta) - 1 > 0$, we have $p(2q-1) \ge p(2(1-\delta)-1) > \delta(2(1-\delta)-1) = 3/32$. Therefore, for any $\varepsilon \le 3/32$ this is not an ε-NE, and so we conclude that $p \le \delta$, as required.

AND gates. For a gate $g = (\mathsf{AND}, u, v, w)$, the player w, who represents the output variable, will play the same bimatrix games as in Section 4.2.2 against u and v, who represent the input variables.

We claim that this gadget will correctly simulate the AND gate for $\delta = 1/8$ and all $\varepsilon \le 1/48$. Let player u play strategy $(1-q_1, q_1)$, let v play strategy $(1-q_2, q_2)$, and let w play strategy $(1-p, p)$, where $q_1, q_2, p \in [0, 1]$. We need to show that in any ε-NE,

(i) if $q_1 \le \delta$ or $q_2 \le \delta$, then $p \le \delta$,

(ii) if $q_1 \ge 1-\delta$ and $q_2 \ge 1-\delta$, then $p \ge 1-\delta$.

The expected payoff of w is $U_w(p) = \frac{1}{2}(1-p)(1-q_1) + \frac{1}{2}(1-p)(1-q_2) + \frac{1}{6}pq_1 + \frac{1}{6}pq_2$ while her expected payoff for playing pure strategy zero is $U_w(0) = \frac{1}{2}(1-q_1) + \frac{1}{2}(1-q_2)$ and for pure strategy one is $U_w(1) = \frac{1}{6}q_1 + \frac{1}{6}q_2$. Then in any ε-NE the following two inequalities must hold.

(a) $U_w(p) \ge U_w(0) - \varepsilon$, or equivalently, $p\left[1 - (q_1+q_2)\frac{2}{3}\right] \le \varepsilon$, and

(b) $U_w(p) \ge U_w(1) - \varepsilon$, or equivalently, $(1-p)\left[1 - (q_1+q_2)\frac{2}{3}\right] \ge -\varepsilon$.

In case (i), suppose that $p > \delta$. Since $q_1 \le \delta$ or $q_2 \le \delta$, we have $p\left[1-(q_1+q_2)\frac{2}{3}\right] \ge p\left[1-(1+\delta)\frac{2}{3}\right] > \delta\left[1-(1+\delta)\frac{2}{3}\right] = 1/32 > 1/48$, where we also used that $1-(1+\delta)\frac{2}{3} > 0$. This contradicts Inequality (a) for any $\varepsilon \le 1/48$. So, $p \le \delta$ as required.

In case (ii), suppose that $p < 1-\delta$. Since $q_1 \ge 1-\delta$ and $q_2 \ge 1-\delta$, we get $(1-p)\left[1-(q_1+q_2)\frac{2}{3}\right] \le (1-p)\left[1-2(1-\delta)\frac{2}{3}\right] < \delta\left[1-2(1-\delta)\frac{2}{3}\right] = -1/48$, where we also used that $1-2(1-\delta)\frac{2}{3} < 0$. This contradicts Inequality (b) for any $\varepsilon \le 1/48$, and so we conclude that $p \ge 1-\delta$ as required.

PURIFY gates. For a gate $g = (\mathsf{PURIFY}, u, v, w)$, the players v and w, who represent the output variables, play the same bimatrix games as in Section 4.2.2 against u, who represents the input variable.

We claim that this gadget will correctly simulate the PURIFY gate for $\delta = 1/8$ and all $\varepsilon \le 1/48$. Let player u play strategy $(1-q, q)$, let v play strategy $(1-p_1, p_1)$, and let w play strategy $(1-p_2, p_2)$, where $q, p_1, p_2 \in [0, 1]$. We need to show that in any ε-NE,

(i) if $q \le \delta$, then $p_1 \le \delta$ and $p_2 \le \delta$,

(ii) if $q \geq 1-\delta$, then $p_1 \geq 1-\delta$ and $p_2 \geq 1-\delta$,

(iii) if $q \in (\delta, 1-\delta)$, either $p_1 \notin (\delta, 1-\delta)$ or $p_2 \notin (\delta, 1-\delta)$.

The expected payoff of v is $U_v(p_1) = \frac{1}{3}(1-p_1)(1-q) + p_1 q$ while her expected payoff for playing pure strategy zero is $U_v(0) = \frac{1}{3}(1-q)$ and for pure strategy one is $U_w(1) = q$. The expected payoff of w is $U_w(p_2) = (1-p_2)(1-q) + \frac{1}{3}p_2 q$ while her expected payoff for playing pure strategy zero is $U_w(0) = 1-q$ and for pure strategy one is $U_w(1) = \frac{1}{3}q$. Then in any ε-NE, we have the following four inequalities that hold.

(a) $U_v(p_1) \geq U_v(0) - \varepsilon$, or equivalently, $p_1\left(\frac{1}{3} - \frac{4}{3}q\right) \leq \varepsilon$, and

(b) $U_v(p_1) \geq U_v(1) - \varepsilon$, or equivalently, $(1-p_1)\left(\frac{1}{3} - \frac{4}{3}q\right) \geq -\varepsilon$.

(c) $U_w(p_2) \geq U_w(0) - \varepsilon$, or equivalently, $p_2\left(1 - \frac{4}{3}q\right) \leq \varepsilon$, and

(d) $U_w(p_2) \geq U_w(1) - \varepsilon$, or equivalently, $(1-p_2)\left(1 - \frac{4}{3}q\right) \geq -\varepsilon$.

In case (i), suppose $p_1 > \delta$. Since $q \leq \delta$ and $\frac{1}{3} - \frac{4}{3}\delta > 0$, we have $p_1\left(\frac{1}{3} - \frac{4}{3}q\right) \geq p_1\left(\frac{1}{3} - \frac{4}{3}\delta\right) > \delta\left(\frac{1}{3} - \frac{4}{3}\delta\right) = \frac{1}{48}$. Hence, for any $\varepsilon \leq 1/48$ this contradicts Inequality (a), and so we conclude that $p_1 \leq \delta$. Now suppose $p_2 > \delta$. Since $q \leq \delta$ and $1 - \frac{4}{3}\delta > 0$, we have $p_2\left(1 - \frac{4}{3}q\right) \geq p_2\left(1 - \frac{4}{3}\delta\right) > \delta\left(1 - \frac{4}{3}\delta\right) = \frac{5}{48}$. So, for any $\varepsilon \leq 5/48$ this contradicts Inequality (c), and we conclude that $p_2 \leq \delta$. Therefore, in any ε-NE, if $q \leq \delta$, we have $p_1 \leq \delta$ and $p_2 \leq \delta$ as required.

In case (ii), suppose that $p_1 < 1-\delta$. Since $q \geq 1-\delta$ and $\frac{1}{3} - \frac{4}{3}(1-\delta) < 0$, we have $(1-p_1)\left(\frac{1}{3} - \frac{4}{3}q\right) \leq (1-p_1)\left(\frac{1}{3} - \frac{4}{3}(1-\delta)\right) < \delta\left(\frac{1}{3} - \frac{4}{3}(1-\delta)\right) = -5/48$. Therefore, for any $\varepsilon \leq 5/48$ this contradicts Inequality (b), and so we conclude that $p_1 \geq 1-\delta$. Now suppose that $p_2 < 1-\delta$. Since $q \geq 1-\delta$ and $1 - \frac{4}{3}(1-\delta) < 0$, we have $(1-p_2)\left(1 - \frac{4}{3}q\right) \leq (1-p_2)\left(1 - \frac{4}{3}(1-\delta)\right) < \delta\left(1 - \frac{4}{3}(1-\delta)\right) = -1/48$. So, for any $\varepsilon \leq 1/48$ this contradicts Inequality (d), and we conclude that $p_2 \geq 1-\delta$. Therefore, in any ε-NE, if $q \geq 1-\delta$, we have $p_1 \geq 1-\delta$ and $p_2 \geq 1-\delta$ as required.

Finally, in case (iii),

- If $q \in (\delta, 1/2)$, suppose that $p_2 > \delta$. Then, we have $p_2\left(1 - \frac{4}{3}q\right) \geq p_2\left(1 - \frac{4}{3}\cdot\frac{1}{2}\right) > \delta\left(1 - \frac{4}{3}\cdot\frac{1}{2}\right) = \frac{1}{24}$, where we used $1 - \frac{4}{3}\cdot\frac{1}{2} > 0$. Therefore, for any $\varepsilon \leq 1/24$ this contradicts Inequality (c), and so we conclude that $p_2 \leq \delta$.

- If $q \in [1/2, 1-\delta)$, suppose that $p_1 < 1-\delta$. Then, we have $(1-p_1)\left(\frac{1}{3} - \frac{4}{3}q\right) \leq (1-p_1)\left(\frac{1}{3} - \frac{4}{3}\cdot\frac{1}{2}\right) < \delta\left(\frac{1}{3} - \frac{4}{3}\cdot\frac{1}{2}\right) = -\frac{1}{24}$, where we used $\frac{1}{3} - \frac{4}{3}\cdot\frac{1}{2} < 0$. Therefore, for any $\varepsilon \leq 1/24$ this contradicts Inequality (b), and so we conclude that $p_1 \geq 1-\delta$.

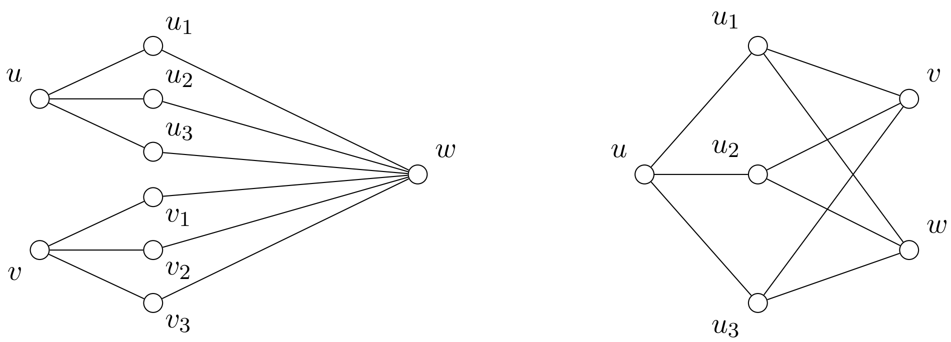
Hence, in any ε-NE, when $q \in (\delta, 1-\delta)$, either $p_1 \notin (\delta, 1-\delta)$ or $p_2 \notin (\delta, 1-\delta)$ as required.

Hardness result. According to the case analysis above, each of the gate constraints from the Pure-Circuit instance are enforced correctly in any ε-NE of the polymatrix game with $\varepsilon \leq 1/48$. Thus, given an ε-NE solution, we can produce a satisfying assignment to the Pure-Circuit instance using the mapping that we defined at the start of the reduction.

Since the construction is the same as the one used for ε-WSNE, we get the same properties that we got there. Namely that no extra normalization is necessary, and that the game is degree 3, bipartite, and has two strategies per player. $\square$

4.2.4 Hardness for Win-Lose Polymatrix games

A polymatrix game is *win-lose* if for every player $i \in [n]$ it holds that the entries of every payoff matrix A_{ij} belong to $\{0, a_i\}$, for some $a_i \in (0, 1]$. In other words, player i either "wins" payoff a_i from the interaction with another player, or "loses" and gets payoff 0. Prior work has



AND gadget PURIFY gadget

Figure 1: The gadgets used for win-lose polymatrix games.

shown that computing an ε-WSNE is PPAD-complete for inverse-polynomial ε via a reduction from GCircuit [LLD21]. In this section we will show that hardness holds for all $\varepsilon < 1/3$, by modifying our hardness result for ε-WSNE given earlier.

Theorem 4.5. *It is* PPAD*-hard to find an ε-WSNE in a win-lose polymatrix game for all $\varepsilon < 1/3$, even in degree 7 bipartite games with two strategies per player.*

Proof. We begin by explaining how to modify the payoff matrices described in Section 4.2.2. Observe that the payoff matrix for NOT gates is already win-lose, so we only need to convert AND gates and PURIFY gates into win-lose payoff matrices. The high level idea for these two gates is to create a number of intermediate players, who all copy the input player's strategy, and whose payoffs to the output player sum to the same payoffs that we used in the ε-WSNE hardness construction. Figure 1 depicts the interaction graphs for these two gates.

AND gates. For a gate $g = (\mathsf{AND}, u, v, w)$, we will create the AND gadget from Figure 1. The player w, that represents the output variable, will play against the auxiliary players u_1, u_2, u_3, and v_1, v_2, v_3 who in turn will "copy" the values, represent the input variables. The payoff matrices for this gadget will be defined as follows, where $X \in \{u_1, v_1\}$, $Y \in \{u_2, u_3, v_2, v_3\}$, and when $Q = v$, then $P \in \{v_1, v_2, v_3\}$ and when $Q = u$, then $P \in \{u_1, u_2, u_3\}$.

w \ X	zero	one
zero	$\frac{1}{6}$, 0	0, 0
one	0, 0	$\frac{1}{6}$, 0

w \ Y	zero	one
zero	$\frac{1}{6}$, 0	0, 0
one	0, 0	0, 0

P \ Q	zero	one
zero	1, 0	0, 0
one	0, 0	1, 0

We claim that this gate will work for all $\varepsilon < 1/3$. Firstly, it is not hard to observe that for the game played between P and Q, in every ε-WSNE of the game: if Q plays zero, then P plays zero; if Q plays one, then P plays one. Hence, we have the following for the gadget.

- If both u and v play one as a pure strategy, then every auxiliary player u_1, u_2, u_3 and v_1, v_2, v_3 plays one. Hence, the payoff to w for playing zero is 0, and the payoff to w for playing one is $1/6 + 1/6 = 1/3$. So, if $\varepsilon < 1/3$, then in all ε-WSNEs the only ε-best response for w is one, as required by the constraints of the AND gate.

- If at least one of u and v play zero as a pure strategy, say that u plays zero, then the auxiliary players u_1, u_2, u_3 will play zero as well. Thus, the payoff to w for playing zero is at least $1/2$, while the payoff to w for playing one is at most $1/6$. So, if $\varepsilon < 1/3$, then in all ε-WSNEs the only ε-best response for w is zero, as required by the constraints of the AND gate.

- The AND gate places no other constraints on the variable w, so we can ignore all other cases, e.g., the case where both u and v play strictly mixed strategies.

PURIFY gates. For a gate $g = (\mathsf{PURIFY}, u, v, w)$, we will create a PURIFY gadget from Figure 1. Players v and w, that represent the output variables, will play against the auxiliary players u_1, u_2, u_3, who in turn will "copy" the values of the input variable that corresponds to player u. The payoff matrices for this gadget will be defined as follows, where $P \in \{u_1, u_2, u_3\}$, $Q \in \{v, w\}$, and $X \in \{u_1, u_2\}$.

v \ X	zero	one
zero	0 0	0 0
one	0 0	0 $\frac{1}{3}$

w \ X	zero	one
zero	0 $\frac{1}{3}$	0 0
one	0 0	0 0

Q \ u_3	zero	one
zero	0 $\frac{1}{3}$	0 0
one	0 0	0 $\frac{1}{3}$

P \ u	zero	one
zero	0 1	0 0
one	0 0	0 1

We claim that this gate will work for all $\varepsilon < 1/3$. Similarly to the AND gadget, it is not hard to observe that for the game played between u and P, in every ε-WSNE of the game: if u plays zero, then P plays zero; if u plays one, then P plays one.

- If u plays zero as a pure strategy, then auxiliary players u_1, u_2, u_3 play zero as well. Thus, strategy zero gives payoff $1/3$ to both v and 1 to w, while strategy one gives payoff 0 to both v and w, so in any ε-WSNE with $\varepsilon < 1/3$ we have that zero is the only ε-best response for both v and w.

- If u plays one as a pure strategy, then auxiliary players u_1, u_2, u_3 play one as well. Thus, strategy one gives payoff 1 to v and and payoff $1/3$ to w, while strategy zero gives payoff 0 to both v and w. So, in any ε-WSNE with $\varepsilon < 1/3$ we have that one is the only ε-best response for v and w.

- Assume now that u mixes between pure strategies zero and one. Then, the auxiliary players u_1, u_2, u_3 might mix between their pure strategies as well; for $i \in \{1, 2, 3\}$ let p_i be the probability player u_i places on pure strategy zero. Then, we have that the following hold.

  - The payoff for v is $\frac{1}{3} \cdot p_3$ for playing zero, and $1 - \frac{1}{3} \cdot (p_1 + p_2 + p_3)$ for playing one.
  - The payoff for w is $\frac{1}{3} \cdot (p_1 + p_2 + p_3)$ for playing zero, and $\frac{1}{3} - \frac{1}{3} \cdot p_3$ for playing one.

  Now assume that in an ε-WSNE, with $\varepsilon < 1/3$, v mixes between zero and one. Then, it must hold that the difference of the payoffs between the two pure strategies is bounded by $1/3$, i.e.,

  $$\left| \frac{1}{3} \cdot p_3 - \left(1 - \frac{1}{3} \cdot (p_1 + p_2 + p_3)\right) \right| < \frac{1}{3}.$$

  Hence, we get that

  $$\frac{2}{3} - \frac{1}{3} \cdot p_3 < \frac{1}{3} \cdot (p_1 + p_2 + p_3).$$

  We claim that in this case w will have to play zero as a pure strategy. Indeed, observe that the payoff of w from zero is $\frac{1}{3} \cdot (p_1 + p_2 + p_3) > \frac{2}{3} - \frac{1}{3} \cdot p_3$, where the inequality follows from above. On the other hand, the payoff from one is $\frac{1}{3} - \frac{1}{3} \cdot p_3$. Hence, the difference between the payoffs of the two pure strategies is larger than $1/3$. Thus, in any ε-WSNE, with $\varepsilon < 1/3$, if v plays a mixed strategy, then zero is the only ε-best response for w.

For the second case, assume that in an ε-WSNE, with $\varepsilon < 1/3$, w mixes between zero and one. Then, it must hold that the difference of the payoffs between the two pure strategies is bounded by $1/3$, i.e.,

$$\left| \frac{1}{3} \cdot (p_1 + p_2 + p_3) - \left(\frac{1}{3} - \frac{1}{3} \cdot p_3 \right) \right| < \frac{1}{3}.$$

Hence, we get that

$$\frac{1}{3} \cdot p_3 < \frac{2}{3} - \frac{1}{3} \cdot (p_1 + p_2 + p_3).$$

We claim that in this case v will have to play one as a pure strategy. Indeed, observe that the payoff of v from zero is $\frac{1}{3} \cdot p_3 < \frac{2}{3} - \frac{1}{3} \cdot (p_1 + p_2 + p_3)$, where the inequality follows from above. On the other hand, the payoff from one is $1 - \frac{1}{3} \cdot (p_1 + p_2 + p_3)$. Hence, the difference between the payoffs of the two pure strategies is larger than $1/3$. Thus, in any ε-WSNE, with $\varepsilon < 1/3$, if w plays a mixed strategy, then one is the only ε-best response for v.

Hardness result. As we have argued above, each of the gate constraints from the PURE-CIRCUIT instance is enforced correctly in any ε-WSNE of the win-lose polymatrix game with $\varepsilon < 1/3$. Thus, given such an ε-WSNE, we can produce a satisfying assignment to the PURE-CIRCUIT instance using the same mapping as we have used before. In addition, all of the games that we have presented are win-lose games and are already normalized, and in addition every player has two strategies.

Now, observe that the reduction described above does not immediately yield a bipartite graph and in addition the maximum degree of the constructed graph is at most nine (this can happen if the vertex that represents the output of an AND gadget, is the input to a different AND gadget as well). However, it is not difficult to tweak the construction and get these two properties too.

Firstly, we perform the following preprocessing step on the PURE-CIRCUIT instance, which results in having a graph with maximum degree seven in the win-lose game constructed here. We replace every node v of the PURE-CIRCUIT instance with three nodes v_1, v_2, v_3 where: v_3 is the output of a NOT gate with input v_2; v_2 in turn is the output of a NOT gate with input v_1; v_3 is the input of the gate(s) that had as input the vertex v; v_1 is the output of the gate that had as output v. It is not difficult to see that this transformation does not change the solutions of the original instance. In addition, observe that under this preprocessing step and Corollary 2.3, the maximum degree of the constructed win-lose game is now seven.

Next, in order to get a bipartite graph we modify the gadget for NOT gates as follows. For a gate $g = (\mathsf{NOT}, u, v)$ we create an auxiliary player u', where u' "copies" the strategy of u and v "negates" the strategy of u'. So, overall v "negates" the strategy of u. To achieve this, the game played between u' and v corresponds to the game constructed for the NOT gates for general polymatrix games and the game between u and u' corresponds to the rightmost game constructed for the AND gate for win-lose polymatrix games. Observe now that the final graph is bipartite: one side contains only the vertices from the preprocessed PURE-CIRCUIT instance and the other side contains only the auxiliary vertices we have created for the gadgets. □

4.3 Threshold Games

Threshold games were introduced by Papadimitriou and Peng [PP21] as an intermediate problem that was used to prove PPAD-hardness for public good games on directed networks. Since then, threshold games have been used to prove PPAD-hardness for throttling equilibria in auction markets [CKK21b] and for the famous Hylland-Zeckhauser scheme [CCPY22].

Threshold Game. A threshold game $\mathcal{G}(V, E)$ is defined on a directed graph $G = (V, E)$. Every node $v \in V$ represents a player with strategy space $\mathbf{x}[v] \in [0, 1]$. For every $v \in V$, we use $N_v := \{u \in V : (u, v) \in E\}$ to denote the set of nodes with outgoing edges towards v. Then, we say that $\mathbf{x} := (\mathbf{x}[v] : v \in V) \in [0, 1]^{|V|}$ is an ε-approximate equilibrium if every $\mathbf{x}[v]$ satisfies

$$\mathbf{x}[v] \in \begin{cases} [0, \varepsilon], & \text{if } \sum_{u \in N_v} \mathbf{x}[u] > 0.5 + \varepsilon; \\ [1 - \varepsilon, 1], & \text{if } \sum_{u \in N_v} \mathbf{x}[u] < 0.5 - \varepsilon; \\ [0, 1], & \text{if } \sum_{u \in N_v} \mathbf{x}[u] \in [0.5 - \varepsilon, 0.5 + \varepsilon]. \end{cases}$$

Definition 6 (ε-ThresholdGameNE)**.** Let $\varepsilon \in [0, 1]$. An instance of the ε-ThresholdGameNE problem consists of a threshold game $\mathcal{G}(V, E)$. The task is to find an ε-approximate equilibrium for $\mathcal{G}(V, E)$.

Papadimitriou and Peng [PP21] proved that there exists a constant $\varepsilon' > 0$ such that ε'-ThresholdGameNE is PPAD-complete, via a reduction from ε-GCircuit, where $\varepsilon' < \varepsilon/5$. Hence, from our hardness result for GCircuit (Theorem 4.1) we obtain that ε-ThresholdGameNE is PPAD-complete for some ε upper bounded by 0.02.

Using a direct reduction from Pure-Circuit we can show that the problem is in fact PPAD-complete for any $\varepsilon < 1/6$. Furthermore, this is tight, since we provide a simple algorithm solving the problem in polynomial time for $\varepsilon = 1/6$. We first present the algorithm achieving the upper bound, and then the improved lower bound through a direct reduction from Pure-Circuit.

4.3.1 Computing a 1/6-Approximate Equilibrium

Theorem 4.6. *The* 1/6-ThresholdGameNE *problem can be solved in polynomial time.*

Proof. Let $G = (V, E)$ be the threshold game graph. The algorithm proceeds as follows.

1. For every node $v \in V$ with in-degree at least two, i.e., $|N_v| \geq 2$, set $\mathbf{x}[v] := 1/6$. Remove all incoming edges of v from the graph. Note that every node in the graph now has in-degree at most one.

2. Repeat until there are no more directed cycles: Pick a directed cycle, and for each node v on the cycle, set $\mathbf{x}[v] := 1/2$. Remove all the edges on the cycle from the graph.

3. Repeat until all nodes have been assigned a value: Since the graph is now acyclic, there exists a node $v \in V$ with unassigned value such that v has in-degree zero, or such that its incoming neighbour $w \in N_v$ has already been assigned a value. If $\sum_{u \in N_v} \mathbf{x}[u] > 0.5 + 1/6$, set $\mathbf{x}[v] := 1/6$. Otherwise, set $\mathbf{x}[v] := 1$.

The algorithm clearly runs in polynomial time. It remains to prove that all nodes satisfy the ε-approximate equilibrium condition for $\varepsilon = 1/6$. Note that when we assign a value to a node, all of its original incoming edges are still present in the graph. Furthermore, we only assign a value to a node once. We proceed by considering the nodes in each step separately.

For nodes which get assigned in Step 3, the 1/6-approximate equilibrium condition is immediately satisfied because of the way in which we pick the value to assign. For a node which gets assigned in Step 2, note that it must have in-degree one, and thus its unique incoming neighbour also lies on the same directed cycle, and also gets assigned value 1/2, which implies that the equilibrium condition is satisfied. Finally, note that the algorithm only assigns values $\geq 1/6$ to nodes. As a result, for any node v which gets assigned in Step 1, we have $\sum_{u \in N_v} \mathbf{x}[u] \geq |N_v|/6 \geq 1/3$, since $|N_v| \geq 2$. In particular, assigning value 1/6 to v satisfies the 1/6-approximate equilibrium condition, because $1/3 \geq 0.5 - 1/6$. □

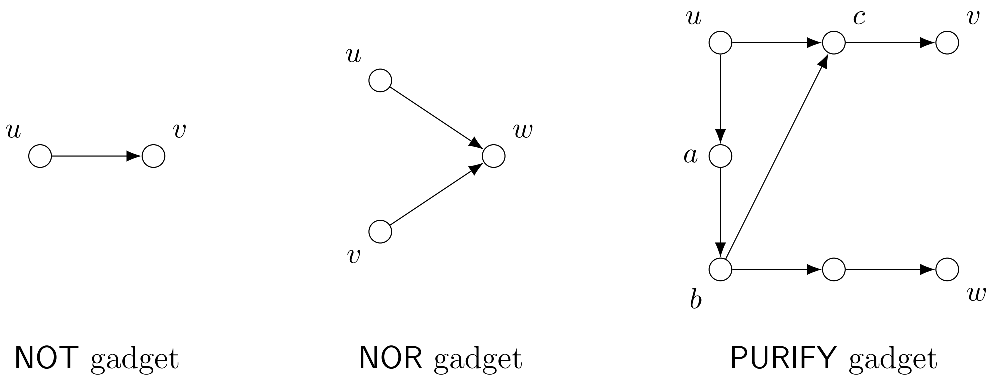


Figure 2: The gadgets used for ThresholdGameNE.

4.3.2 Hardness of ε-Approximate Equilibrium for any $\varepsilon < 1/6$

Theorem 4.7. *ε-ThresholdGameNE is PPAD-complete for every $\varepsilon < 1/6$, even when the in- and out-degree of each node is at most two.*

Proof. In order to prove that ε-ThresholdGameNE is PPAD-hard for $\varepsilon < 1/6$, we will reduce from Pure-Circuit that uses the gates NOT, NOR, and PURIFY (we include the NOT gate because we will later make use of Corollary 2.3 to argue about the degrees of nodes). We will encode 0 values in the Pure-Circuit problem as values in the range $[0, \varepsilon]$ in ThresholdGameNE, while 1 values will be encoded as values in the range $[1 - \varepsilon, 1]$. Then, each gate of Pure-Circuit will be simulated by the corresponding gadget depicted in Figure 2.

NOT gates. A (NOT, u, v) gate will be simulated by the NOT gadget depicted in Figure 2. We argue that the gadget works for any $\varepsilon < 0.25$, and thus in particular for any $\varepsilon < 1/6$.

- If $\mathbf{x}[u] \leq \varepsilon < 0.5 - \varepsilon$, then according to the definition of threshold games in any ε-approximate equilibrium it must hold that $\mathbf{x}[v] \geq 1 - \varepsilon$. Hence, when the input encodes a 0, the output value will encode a 1.
- If $\mathbf{x}[u] \geq 1 - \varepsilon > 0.5 + \varepsilon$, then according to the definition of threshold games in any ε-approximate equilibrium it must hold that $\mathbf{x}[v] \leq \varepsilon$. Hence, when the input encodes a 1, the output value will encode a 0.

NOR gates. A (NOR, u, v, w) gate will be simulated by the NOR gadget depicted in Figure 2. We claim that the gadget works for any $\varepsilon < 1/6$.

- If both $\mathbf{x}[u]$ and $\mathbf{x}[v]$ are in $[0, \varepsilon]$, then $\mathbf{x}[u] + \mathbf{x}[v] \leq 2\varepsilon < 0.5 - \varepsilon$. Hence, in any ε-approximate equilibrium it must hold that $\mathbf{x}[w] \geq 1 - \varepsilon$. In other words, when both input values encode 0s, then the output value of the NOR gadget will encode 1.
- If at least one of $\mathbf{x}[u]$ and $\mathbf{x}[v]$ is in $[1 - \varepsilon, 1]$, then $\mathbf{x}[u] + \mathbf{x}[v] \geq 1 - \varepsilon > 0.5 + \varepsilon$. Hence, in any ε-approximate equilibrium it must hold that $\mathbf{x}[w] \leq \varepsilon$. In other words, when at least one input value encodes a 1, then the output value of the NOR gadget will encode 1.

PURIFY gates. A $(\mathsf{PURIFY}, u, v, w)$ gate will be simulated by the PURIFY gadget depicted in Figure 2. We argue that the gadget works for any $\varepsilon < 1/6$.

- If $\mathbf{x}[u] \leq \varepsilon$, then, in particular, $\mathbf{x}[u] < 0.5 - \varepsilon$, and thus $\mathbf{x}[a] \geq 1 - \varepsilon$. Then, since $\mathbf{x}[a] \geq 1 - \varepsilon > 0.5 + \varepsilon$, it follows that $\mathbf{x}[b] \leq \varepsilon$. By applying the same arguments again, we obtain that $\mathbf{x}[w] \leq \varepsilon$. Furthermore, since $\mathbf{x}[u] + \mathbf{x}[b] \leq 2\varepsilon < 0.5 - \varepsilon$, it follows that

$\mathbf{x}[c] \geq 1 - \varepsilon$, and thus $\mathbf{x}[v] \leq \varepsilon$. In other words, when the input value encodes a 0, then both output values of the PURIFY gadget will encode 0s.

- If $\mathbf{x}[u] \geq 1 - \varepsilon$, then it is not hard to see that $\mathbf{x}[b] \geq 1 - \varepsilon$, and thus $\mathbf{x}[w] \geq 1 - \varepsilon$. In addition, since $\mathbf{x}[u] + \mathbf{x}[b] \geq 2(1 - \varepsilon) > 0.5 + \varepsilon$, it follows that $\mathbf{x}[c] \leq \varepsilon$, and thus $\mathbf{x}[v] \geq 1 - \varepsilon$. In other words, when the input value encodes a 1, then both output values of the PURIFY gadget will encode 1s.

- Finally, it remains to prove that no matter what the input u is, at least one of the two output values v, w of the PURIFY gadget will encode a pure bit value. Assume that w does not encode a pure bit value, i.e., $\mathbf{x}[w] \in (\varepsilon, 1 - \varepsilon)$. We will show that in that case it must be that $\mathbf{x}[v] \notin (\varepsilon, 1 - \varepsilon)$. Since $\mathbf{x}[w] \in (\varepsilon, 1 - \varepsilon)$, it follows that $\mathbf{x}[b] \in [0.5 - \varepsilon, 0.5 + \varepsilon]$. Similarly, since $\mathbf{x}[b] \in [0.5 - \varepsilon, 0.5 + \varepsilon] \subseteq (\varepsilon, 1 - \varepsilon)$, it also follows that $\mathbf{x}[u] \in [0.5 - \varepsilon, 0.5 + \varepsilon]$. As a result, we obtain that $\mathbf{x}[u] + \mathbf{x}[b] \geq 2(0.5 - \varepsilon) > 0.5 + \varepsilon$, which implies that $\mathbf{x}[c] \leq \varepsilon$, and thus $\mathbf{x}[v] \geq 1 - \varepsilon$, as desired.

The reduction. Given a Pure-Circuit instance over variables V, we produce a ThresholdGameNE instance by replacing each gate with the corresponding gadget depicted in Figure 2. Then, given a solution $\mathbf{x}$ to the ε-ThresholdGameNE instance with $\varepsilon < 1/6$, we can produce a solution $\mathbf{x'}$ for Pure-Circuit as follows. For every $v \in V$:

- if $\mathbf{x}[v] \leq \varepsilon$, then we set $\mathbf{x'}[v] = 0$;
- if $\mathbf{x}[v] \geq 1 - \varepsilon$, then we set $\mathbf{x'}[v] = 1$;
- else we set $\mathbf{x'}[v] = \perp$.

The correctness of the reduction follows from the arguments above. Finally, by using Corollary 2.3, the constructed instances of ThresholdGameNE will all satisfy that the in- and out-degree of every node is at most two. Indeed, the new "auxiliary" nodes introduced by the PURIFY gadget all satisfy this, and the contribution of each gadget to the in- and out-degree of "original" nodes is exactly the same as the contribution of the corresponding Pure-Circuit gates in the interaction graph. □

A On the Definition of Pure-Circuit

In this section, we explore different ways to weaken the definition of Pure-Circuit, and show how, in each case, the problem is no longer PPAD-hard.

No PURIFY gate. If we allow all gates, except the PURIFY gate (so only NOT, COPY, OR, AND, NOR, NAND), then the problem becomes polynomial-time solvable. Indeed, it suffices to assign value $\perp$ to all the nodes.

No Negation. If we allow all gates, except the ones that perform some kind of negation (so only PURIFY, COPY, OR, AND), then the problem becomes polynomial-time solvable. Indeed, assigning the value 1 to each node (or, alternatively, the value 0 to each node) always yields a solution. More generally, we can make the following observation: for any set of gates that can be implemented by monotone functions, the problem lies in the class PLS, and is thus unlikely to be PPAD-complete. Indeed, as already mentioned in Section 2, we can view any Pure-Circuit instance with n nodes as a function $F : [0,1]^n \to [0,1]^n$, where each gate is replaced by a continuous function that is consistent with the gate-constraint. Then any fixed point of F yields a solution to the Pure-Circuit instance. If each of the gates can be replaced by a continuous *monotone* function, then the problem of finding a fixed point of F is an instance of Tarski's fixed point theorem, which is known to lie in PLS [EPRY20].

No robustness. If we allow all gates (PURIFY, NOT, COPY, OR, AND, NOR, NAND), but we drop the robustness requirement from the logical gates OR, AND, NOR, NAND, then the problem can be solved in polynomial time.

Construct the interaction graph G of the Pure-Circuit instance, as defined in Section 2.1. In the first stage of the algorithm, as long as there exists a directed cycle in graph G, we do the following:

1. Pick an arbitrary directed simple cycle of G.
2. For each node u on the simple cycle C, assign value $\perp$ to it, i.e., $\mathbf{x}[u] := \perp$.
3. Remove all nodes on the simple cycle C from the graph G, including all their incident edges.

At the end of this procedure, G no longer contains any cycles. In the second stage of the algorithm we then repeat the following, until G is empty:

1. Pick any source u of G (which must exist, since G contains no cycles).
2. Let g be the (unique) gate that has u as output. Since u does not have incoming edges in G, all inputs of g have already been assigned a value. If u is the only output of g, then assign a value to u that satisfies the gate, and remove u and its edges from G. If the gate g also has another output v, then g is a PURIFY gate, and there are two cases:
   - If v has not been assigned a value yet, then assign values to both u and v such that the gate is satisfied. Remove u and v and their edges from G.
   - If v has already been assigned a value, then this happened in the first stage of the algorithm, and both v and the input to g were assigned value $\perp$ (if v lies on a simple cycle C, then so does the input of gate g, because v has a single incoming edge in the original interaction graph). In that case, we assign value 0 or 1 to u and g is satisfied. Remove u and its edges from G (v was already removed in the first stage).

Since a node is removed from G only when it is assigned a value, all nodes have been assigned a value at the end of the algorithm. We argue that all gates are satisfied. Clearly, any gate that has an output node that is still present in G after the end of the first stage will be satisfied (by construction of the second stage). Thus, it remains to consider any gate g such that all its output nodes are removed in the first stage. There are three cases:

- g is a NOT or COPY gate: if the output lies on a simple cycle C, then so does the input. Both are thus assigned value $\perp$ and the gate is satisfied.
- g is a (non-robust) OR, AND, NOR, or NAND gate: if the output lies on a simple cycle C, then so does at least one of its inputs. Thus, at least one input is also assigned value $\perp$, and the gate is satisfied. Note that we crucially used the non-robustness of the gate here.
- g is a PURIFY gate: if an output v of g lies on a simple cycle C, then the input u of g also lies on C, and so both are removed at the same time from G. However, the other output w of g cannot lie on that same simple cycle C, and after u is removed, w does not have an incoming edge anymore and will thus not be removed in the first stage. Thus, g cannot be a PURIFY gate.

Adding Constant Gates. It is a natural idea to try to add constant gates in an attempt to make the problem hard for a set of gates for which the problem is not PPAD-hard. By constant gates, we mean a 0-gate which has one output and no input, and which enforces that the value of its output node always be 0, and a 1-gate defined analogously. No matter which subset of gates $S \subseteq \{\mathsf{PURIFY}, \mathsf{NOT}, \mathsf{COPY}, \mathsf{OR}, \mathsf{AND}, \mathsf{NOR}, \mathsf{NAND}\}$ we use, adding the constant gates 0 and 1 does not change the complexity of the problem. Indeed, it is easy to see that the constants can be "propagated" through the circuit. If a constant is an input to a PURIFY, NOT, or COPY gate, then we can replace the output(s) of that gate by constants that satisfy the gate-constraint. If a constant is an input to an X gate, where $\mathsf{X} \in \{\mathsf{OR}, \mathsf{AND}, \mathsf{NOR}, \mathsf{NAND}\}$, then there are two cases: either we can replace the output by a constant, or we can replace the gate by an X gate with the same input twice (namely, the other input). In order to create a copy of the other input, we can either use the COPY gate, or the NOT gate, or the PURIFY gate. If none of these three gates lies in S, then PURE-CIRCUIT with gates $S \cup \{0, 1\}$ is polynomial-time solvable, since it is polynomial-time solvable with gates $S \cup \{\mathsf{NOT}, 0, 1\}$ (by the propagation argument, and the lack of PURIFY gate).

B Proof of Corollary 2.3

Proof. Let X and Y be as required by the statement of Corollary 2.3. By Corollary 2.2 it immediately follows that PURE-CIRCUIT with gates $\{\mathsf{PURIFY}, \mathsf{X}, \mathsf{Y}\}$ is PPAD-complete. Now consider an instance of PURE-CIRCUIT with gates $\{\mathsf{PURIFY}, \mathsf{X}, \mathsf{Y}\}$. We will explain how to turn it into an instance that satisfies the three conditions of Corollary 2.3. We proceed in three steps, where each step adds more structure to the instance, without destroying any of the structure introduced in a previous step.

Step 1. First of all, for every node u that is used as an input to k gates $g_1, \dots, g_k$, we construct a binary tree consisting of PURIFY gates that is rooted at u and has leaves $u_1, \dots, u_k$. Then, we modify each gate g_i so that it uses u_i as input instead of u. As a result, the instance now satisfies that every node is the input of at most one gate. Note that for any solution $\mathbf{x}$, if $\mathbf{x}[u] \in \{0, 1\}$, then $\mathbf{x}[u_i] = \mathbf{x}[u]$ for all i. On the other hand, if $\mathbf{x}[u] = \perp$, then the $\mathbf{x}[u_i]$ could have different values. Despite this, we argue that this is a valid reduction, namely that restricting $\mathbf{x}$ to the original nodes yields a solution to the original instance. To see this, it suffices to notice that, no matter what the type T of gate g_i is, if u satisfies the condition on the left of the "$\implies$" sign in the definition of T, then the condition is still satisfied if we replace u by u_i.

Step 2. After the first step, the instance now satisfies that every node is the input of at most one gate, in particular $(d_{in}, d_{out}) \in \{1,2\} \times \{0,1,2\}$. In the next step, we will further enforce that $(d_{in}, d_{out}) \neq (2,2)$ and make the interaction graph bipartite. To do this, we replace every node u by a small gadget that consists of nodes u_a, u_b, and some other auxiliary nodes. If u was the output of some gate g, then g now has output u_a instead. Similarly, if u was the input of some gate g', then g' has input u_b instead. The gadget will ensure that $\mathbf{x}[u_a] \in \{0,1\} \implies \mathbf{x}[u_b] = \mathbf{x}[u_a]$. Thus, by the same argument as in the previous paragraph, restricting $\mathbf{x}$ to $\{u_a : u \in V\}$ yields a solution to the original instance. In the case $\mathsf{X} = \mathsf{COPY}$, the gadget is simply implemented by using a COPY gate with input u_a and output u_b, i.e., a $\mathsf{COPY}(u_a, u_b)$ gate. Applying this transformation to every node u achieves two things: (i) the interaction graph only contains nodes with $(d_{in}, d_{out}) \in (\{1,2\} \times \{0,1,2\}) \setminus \{(2,2)\}$, (ii) the interaction graph is bipartite ($A = \{u_a : u \in V\}$, $B = \{u_b : u \in V\}$). In the case $\mathsf{X} = \mathsf{NOT}$, the gadget uses auxiliary (new) nodes v, w, w' and consists of the gates $\mathsf{NOT}(u_a, v)$, $\mathsf{PURIFY}(v, w, w')$, and $\mathsf{NOT}(w, u_b)$. It is easy to check that this gadget satisfies the same properties as before, in particular (i) and (ii). For (ii), note that we add nodes w, w' to set A and node v to set B to make the graph bipartite.

Step 3. It remains to enforce that every node is the input of *exactly* one gate. Currently, this only holds with "at most" instead of "exactly". We begin by introducing a new node v^*. Note that v^* is not used as an input of any gate, and it is also the only node that is not used as an output by any gate. Let V_{sink} be the set of all nodes that are not used as an input by any gate. In particular, $v^* \in V_{sink}$. Using a binary tree of Y-gates which takes the nodes in V_{sink} as inputs, we can ensure that the root u_{sink} of the binary tree is the only node that is not used as input by any gate. By using additional X-gates at the leaves (when needed) we can ensure that the graph remains bipartite. Note that v^* is still the only node that is not used as an output of any gate. We finish the construction by adding an X-gate with input u_{sink} and output v^*. If that makes the graph non-bipartite, then we can instead introduce a new node v and gates $\mathsf{X}(u_{sink}, v)$ and $\mathsf{X}(v, v^*)$ instead. The instance now satisfies all three conditions of the statement. □

C Bimatrix Games

In this section we show lower bounds for finding *relative* approximate well-supported equilibria in bimatrix games. Unlike the *additive* approximations we have studied so far, in a relative approximate equilibrium ε is not measuring the difference between the current strategy's payoff and a deviation's payoff, but instead, the *ratio* of that difference over the deviation's payoff.

Daskalakis [Das13] proved that computing a relative ε-WSNE in bimatrix games with payoffs in $[-1,1]$ is PPAD-complete for any constant $\varepsilon \in [0,1)$. This left open the question of inapproximability of bimatrix games with non-negative payoffs[1]. It is known that for bimatrix games with payoffs in $[0,1]$ there is a polynomial time algorithm for finding a relative 1/2-WSNE [FNS07]. The best hardness result for games with payoffs in $[0,1]$ was given by Rubinstein [Rub18], who showed that there exists a constant ε such that relative ε-WSNE is PPAD-hard.

Here we show that the problem is hard for any $\varepsilon \leq 1/57$. A bimatrix game is a special case of a polymatrix game (see Section 4.2) where we only have two players connected by an edge. Hence, we use the same notation as that of the aforementioned section. Before presenting this section's main result, let us define the equilibrium notion that will be studied in this section.

Definition 7 (relative ε-WSNE)**.** The strategy profile $\mathbf{s}$ is a relative ε-well-supported Nash equilibrium (relative ε-WSNE) if for every player, each action in the support of her strategy is at least as good as any other action up to a fraction ε of the latter action's absolute expected

[1]Relative ε-WSNE (as well as relative ε-NE) are scale invariant. Therefore, multiplying all payoffs by a positive constant does not affect the equilibria for a given ε.

utility. In particular, for non-negative utilities we have

$$\forall i \in [n], \forall k \in \mathsf{supp}(s_i), \quad u_i(k, \mathbf{s}_{-i}) \geq (1-\varepsilon) \cdot \mathrm{br}_i(\mathbf{s}_{-i}).$$

Theorem C.1. *Computing a relative ε-WSNE in a bimatrix game with non-negative payoffs is* PPAD*-complete for any $\varepsilon \leq 1/57$.*

The remainder of this section is devoted to the proof of this theorem.

Hardness. We will reduce from the PPAD-complete problem of computing an (additive) ε'-WSNE in a polymatrix game with $2n$ players (nodes) and two actions per player for $\varepsilon' < 1/3$ (as shown in Theorem 4.3) to the problem of computing a relative ε-WSNE in a $2n \times 2n$ bimatrix game for $\varepsilon = \varepsilon'/\beta$, where $\beta = 18.9$. Our reduction closely follows that of Rubinstein's [Rub18], but instead of reducing from (additive) ε'-NE in polymatrix games, we reduce from their ε'-WSNE counterpart for which we obtained hardness for significantly higher ε'.

The construction of the bimatrix game is identical to that of the aforementioned paper, but we optimize the involved parameters so that we get the highest ε possible. In particular, since again we reduce from a *bipartite* polymatrix game (see Theorem 4.3) we consider the two sides of the graph, namely the R-side and the C-side. Without loss of generality, we consider that each side of the graph has n nodes, since we can add "dummy" nodes to the smallest of the two sides. We then consider two super-players, namely the *row player* R and the *column player* C for the two sets of nodes. Each super-player "represents" her nodes in the new bimatrix game, meaning that her action set consists of the union of the individual nodes' action sets. We embed the polymatrix game into the bimatrix game, and multiply all the corresponding payoffs with a positive constant $\lambda < 1$ (to be defined later) so that all payoffs are normalized in $[0, \lambda]$. Then, to this game we add an "imitation game" as follows. To the payoff matrix of the row player we add a "block identity matrix" $\mathbb{1}_{b(2\times 2)}$ which is a $2n \times 2n$ matrix with its diagonal 2×2 blocks having 1's and the rest 0's. To the payoff matrix of the column player we add a modified "block identity matrix" which is like $\mathbb{1}_{b(2\times 2)}$ but with its entries shifted by two columns to the right (modulo $2n$).

For simplicity, given a strategy of a player in the bimatrix game, we refer to the total probability mass of the two actions corresponding to the polymatrix node $i \in [n]$ as the *mass of node i* and we denote it as $x(i), y(i)$ for the row and column player, respectively. The actions 0, 1 of node i and their probability masses are denoted by $(i : a)$, and $x(i : a)$, $y(i : a)$, respectively, for $a \in \{0, 1\}$. For ease of presentation in what follows, we will refer to the players of the polymatrix game as "nodes" and reserve the word "players" for the bimatrix game's participants. We denote by $N_R(i)$ the neighbourhood of node i that belongs to the R-side of the polymatrix game (and respectively $N_C(i)$ for a node i on the C-side).

First, we present modified versions of the respective results of [Rub18] tailored to the needs of our reduction. Since their proofs are almost identical to those of the aforementioned paper, we have preserved most of their notation.

Lemma C.2 (Modified Lemma 9.3 of [Rub18])**.** *In every (x, y) relative ε-WSNE, for $\lambda < \frac{1-\varepsilon}{2}$, $x(i), y(i) \in \left[\frac{1-\varepsilon-2\lambda}{n}, \frac{(1-\varepsilon-2\lambda)^{-1}}{n}\right]$.*

Proof. Let $t_R = \max_i x(i)$ and $i^*_R = \arg\max_i x(i)$, and let us call $U_R((i : a), y)$ the (expected) payoff that the row player gets from playing action $(i : a)$, $a \in \{0, 1\}$ (respectively, $U_C(x, (i : a))$ for the column player). We will show that, for any $j \in [n]$, if $x(j) < (1 - \varepsilon - 2\lambda)t_R$ then $y(j + 1) = 0$. For any action $(j + 1 : a)$ that the column player picks, her payoff is at most $U_C(x, (j + 1 : a)) \leq x(j) + 2 \cdot \lambda \cdot t_R$. That is due to the imitation game that gives her payoff at most $x(j)$ and the payoffs induced due to her neighbours $k \in N(j + 1)$ in the polymatrix game that give her positive payoff[2], combined with the fact that $x(k) \leq t_R$. Therefore, $U_C(x,(j+1:a)) < (1-\varepsilon)t_R$, but the column player can guarantee a payoff of at least t_R by playing action $(i^*_R+1:a)$ for any $a \in \{0,1\}$. This would contradict the ε-WSNE condition, therefore $y(j+1)=0$. Similarly, for any $j \in [n]$, if $y(j) < (1-\varepsilon-2\lambda)t_C$ then $x(j)=0$, where $t_C = \max_i y(i)$.

Now we claim that for every $i \in [n]$, $x(i), y(i) > 0$. For the sake of contradiction assume that $x(i)=0$ without loss of generality. Then, since $\lambda < (1-\varepsilon)/2$ and $t_R \geq 1/n > 0$, we get $x(i) < (1-\varepsilon-2\lambda)t_R$, and therefore $y(i+1)=0$. With a similar argument for $y(i+1)$ we deduce from the previous paragraph that $x(i+1)=0$, and this inductively yields that for all $i \in [n]$, $x(i)=y(i)=0$, a contradiction. Therefore, for all $i \in [n]$, $x(i), y(i) > 0$. This, combined with the above paragraph yields that for all $i \in [n]$, $x(i) \geq (1-\varepsilon-2\lambda)t_R \geq (1-\varepsilon-2\lambda)/n$.

The upper bound is deduced by the fact that if $t_R > (1-\varepsilon-2\lambda)^{-1}/n$. Since there exists a node i with $x(i) \leq 1/n$ (otherwise $\sum_{j\in[n]} x(j) > 1$), we have that $x(i) < (1-\varepsilon-2\lambda)t_R$, which cannot happen as shown in the first paragraph of the proof. Therefore, for any $i \in [n]$, $x(i) \leq t_R \leq (1-\varepsilon-2\lambda)^{-1}/n$. A similar argument holds for t_C which yields the upper bound for $y(i)$. □

Corollary C.3 (Modified Corollary 9.4 of [Rub18])**.** *In every (x,y) relative ε-WSNE, for $\lambda < \frac{1-\varepsilon}{2}$, the expected utilities of both players for playing any action against the strategy of the other player are in $\left[\frac{1-\varepsilon-2\lambda}{n}, \frac{(1+2\lambda)(1-\varepsilon-2\lambda)^{-1}}{n}\right]$.*

Proof. The lower bound comes from the lower bound of the above lemma and the fact that the row player from any action $(i:a)$, $a \in \{0,1\}$ against y, gets at least the payoff of the imitation game, multiplied by $y(i:a)+y(i:1-a)=y(i)$. The upper bound comes from the upper bound of the above lemma and the fact that the row player from any action $(i:a)$, $a \in \{0,1\}$ against y gets from each entry $((i:a),j)$, $j \in [n]$ of the matrix at most payoff $1 \cdot y(i)$ from the imitation game, and additional payoff of at most $\lambda \cdot y(i)$ from each of two neighbouring nodes in the polymatrix game (note again that only two out of maximum three neighbours can give her positive payoff). A symmetric argument holds for the column player's bounds. □

Fix $\beta = 18.9$. Now we will show that, given an $\varepsilon' < 1/3$, for any (x,y) which is a relative ε'/β-WSNE in the constructed bimatrix game, the marginal distributions of the actions $(i:a)$, $a \in \{0,1\}$ for each node $i \in [n]$ constitute an (additive) ε'-WSNE of the original polymatrix game. In particular, we will prove that the strategy profile where each node i in the polymatrix game belonging to the R-side of the bipartite graph plays $(x(i:0)/x(i), x(i:1)/x(i))$, and similarly each node j of the C side of the bipartite graph plays $(y(j:0)/y(j), y(j:1)/y(j))$ is an ε'-WSNE.

For the sake of contradiction, assume that the marginal distributions of the node strategies are not such an ε'-WSNE. Then, given the marginals induced by (x,y) there is a node i on the left side of the bipartite graph (without loss of generality) for whom playing action $(i:a)$ for some $a \in \{0,1\}$ against y gives her additive ε' more expected utility than what the action $(i:1-a)$ gives. This discrepancy, translated in the bimatrix game (where we have multiplied all payoffs of the polymatrix game by λ) means that the difference in payoff would be at least $\varepsilon' \cdot \lambda \cdot \frac{1-\varepsilon-2\lambda}{n}$ due to Corollary C.3, where $\varepsilon = \varepsilon'/\beta$. Then, again by Corollary C.3 which bounds her maximum utility from playing any action $(i:a)$ against the mixed strategy y, we conclude that her relative increase in expected payoff is at least

$$\frac{u_R((i:1-a),y) + \varepsilon' \cdot \lambda \cdot \frac{1-\varepsilon-2\lambda}{n}}{u_R((i:1-a),y)} \geq 1 + \frac{\varepsilon' \cdot \lambda \cdot \frac{1-\varepsilon-2\lambda}{n}}{\frac{(1+2\lambda)(1-\varepsilon-2\lambda)^{-1}}{n}} \quad \text{(by Corollary C.3)}$$

[2]From the PPAD-hard ε-WSNE instances we construct for polymatrix games in Section 4.2.2, observe that at most two neighbours (out of three) give positive payoff to the node. Namely, only its parents can give it positive payoff while its children always give it 0 payoff.

$$= 1 + \varepsilon' \cdot \frac{\lambda}{1+2\lambda} \cdot (1-\varepsilon-2\lambda)^2$$
$$= 1 + \beta \cdot \varepsilon \cdot \frac{\lambda}{1+2\lambda} \cdot (1-\varepsilon-2\lambda)^2 \quad \text{(since } \varepsilon = \varepsilon'/\beta\text{)}.$$

We now find the value for λ that maximizes the above expression, namely, $\lambda = \frac{-3+\sqrt{17-8\varepsilon}}{8}$. For this value of λ and for every $\varepsilon = \varepsilon'/\beta \leq 1/57$ we have that

$$1 + \beta \cdot \varepsilon \cdot \frac{\lambda}{1+2\lambda} \cdot (1-\varepsilon-2\lambda)^2 > \frac{1}{1-\varepsilon}$$

Note that the optimum value of λ is potentially irrational, and therefore, not suitable as input to the algorithm of our reduction. Nevertheless, the proof goes through for $\lambda = 0.1383$. The above strict inequality shows that the action that the node deviated to gives the bimatrix player more than relative $1/(1-\varepsilon)$ expected utility, and therefore by definition our initial profile (x, y) is not a relative ε-WSNE (for $\varepsilon = \varepsilon'/\beta$), a contradiction. This completes the proof of Theorem C.1.

Remark 3. In fact, Theorem C.1 holds for any $\varepsilon < \frac{1}{3\beta^*}$, where β^* is the optimum value from the above proof. This β^* is in $(18.86, 18.87)$, but we have picked 18.9 for ease of presentation.

Acknowledgements

We thank the anonymous reviewers for comments and suggestions that helped improve the presentation of the paper. The second author was supported by EPSRC grant EP/W014750/1 "New Techniques for Resolving Boundary Problems in Total Search".

References

[CCPY22] Thomas Chen, Xi Chen, Binghui Peng, and Mihalis Yannakakis. Computational hardness of the Hylland-Zeckhauser scheme. In *Proceedings of the 33rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2253–2268, 2022. doi:10.1137/1.9781611977073.90.

[CD09] Xi Chen and Xiaotie Deng. On the complexity of 2D discrete fixed point problem. *Theoretical Computer Science*, 410(44):4448–4456, 2009. doi:10.1016/j.tcs.2009.07.052.

[CDDT09] Xi Chen, Decheng Dai, Ye Du, and Shang-Hua Teng. Settling the complexity of Arrow-Debreu equilibria in markets with additively separable utilities. In *Proceedings of the 50th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 273–282, 2009. doi:10.1109/FOCS.2009.29.

[CDO15] Xi Chen, David Durfee, and Anthi Orfanou. On the complexity of Nash equilibria in anonymous games. In *Proceedings of the 47th ACM Symposium on Theory of Computing (STOC)*, pages 381–390, 2015. doi:10.1145/2746539.2746571.

[CDT09] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM*, 56(3):14:1–14:57, 2009. doi:10.1145/1516512.1516516.

[CKK21a] Xi Chen, Christian Kroer, and Rachitesh Kumar. The complexity of pacing for second-price auctions. In *Proceedings of the 22nd ACM Conference on Economics and Computation (EC)*, page 318, 2021. doi:10.1145/3465456.3467578.

[CKK21b] Xi Chen, Christian Kroer, and Rachitesh Kumar. Throttling equilibria in auction markets. In *Proceedings of the 17th International Conference on Web and Internet Economics (WINE)*, page 551, 2021. doi:10.1007/978-3-030-94676-0.

[CPY17] Xi Chen, Dimitris Paparas, and Mihalis Yannakakis. The complexity of non-monotone markets. *Journal of the ACM*, 64(3):20:1–20:56, 2017. doi:10.1145/3064810.

[CSVY08] Bruno Codenotti, Amin Saberi, Kasturi Varadarajan, and Yinyu Ye. The complexity of equilibria: Hardness results for economies via a correspondence with games. *Theoretical Computer Science*, 408(2–3):188–198, 2008. doi:10.1016/j.tcs.2008.08.007.

[Das13] Constantinos Daskalakis. On the complexity of approximating a Nash equilibrium. *ACM Transactions on Algorithms*, 9(3):1–35, 2013. doi:10.1145/2483699.2483703.

[DFHM22] Argyrios Deligkas, John Fearnley, Alexandros Hollender, and Themistoklis Melissourgos. Constant inapproximability for PPA. In *Proceedings of the 54th ACM Symposium on Theory of Computing (STOC)*, pages 1010–1023, 2022. doi:10.1145/3519935.3520079.

[DFS20] Argyrios Deligkas, John Fearnley, and Rahul Savani. Tree polymatrix games are PPAD-hard. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 38:1–38:14, 2020. doi:10.4230/LIPICS.ICALP.2020.38.

[DFSS17] Argyrios Deligkas, John Fearnley, Rahul Savani, and Paul Spirakis. Computing approximate Nash equilibria in polymatrix games. *Algorithmica*, 77(2):487–514, 2017. doi:10.1007/s00453-015-0078-7.

[DGP09] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009. doi:10.1137/070699652.

[DQS12] Xiaotie Deng, Qi Qi, and Amin Saberi. Algorithmic solutions for envy-free cake cutting. *Operations Research*, 60(6):1461–1476, 2012. doi:10.1287/opre.1120.1116.

[DSZ21] Constantinos Daskalakis, Stratis Skoulakis, and Manolis Zampetakis. The complexity of constrained min-max optimization. In *Proceedings of the 53rd ACM Symposium on Theory of Computing (STOC)*, pages 1466–1478, 2021. doi:10.1145/3406325.3451125.

[EPRY20] Kousha Etessami, Christos Papadimitriou, Aviad Rubinstein, and Mihalis Yannakakis. Tarski's theorem, supermodular games, and the complexity of equilibria. In *Proceedings of the 11th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 18:1–18:19, 2020. doi:10.4230/LIPIcs.ITCS.2020.18.

[EY10] Kousha Etessami and Mihalis Yannakakis. On the complexity of Nash equilibria and other fixed points. *SIAM Journal on Computing*, 39(6):2531–2597, 2010. doi:10.1137/080720826.

[FNS07] Tomas Feder, Hamid Nazerzadeh, and Amin Saberi. Approximating Nash equilibria using small-support strategies. In *Proceedings of the 8th ACM Conference on Electronic Commerce (EC)*, pages 352–354, 2007. doi:10.1145/1250910.1250961.

[FRFGZ18] Aris Filos-Ratsikas, Søren Kristoffer Still Frederiksen, Paul W. Goldberg, and Jie Zhang. Hardness results for Consensus-Halving. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 24:1–24:16, 2018. doi:10.4230/LIPIcs.MFCS.2018.24.

[FRGH+21] Aris Filos-Ratsikas, Yiannis Giannakopoulos, Alexandros Hollender, Philip Lazos, and Diogo Poças. On the complexity of equilibrium computation in first-price auctions. In *Proceedings of the 22nd ACM Conference on Economics and Computation (EC)*, pages 454–476, 2021. doi:10.1145/3465456.3467627.

[GH21] Paul W. Goldberg and Alexandros Hollender. The Hairy Ball problem is PPAD-complete. *Journal of Computer and System Sciences*, 122:34–62, 2021. doi:10.1016/j.jcss.2021.05.004.

[GHI+22] Paul W. Goldberg, Alexandros Hollender, Ayumi Igarashi, Pasin Manurangsi, and Warut Suksompong. Consensus halving for sets of items. *Mathematics of Operations Research*, 2022. doi:10.1287/moor.2021.1249.

[Jan68] Elena Janovskaja. Equilibrium points in polymatrix games. *Lithuanian Mathematical Journal*, 8(2):381–384, 1968. doi:10.15388/lmj.1968.20224.

[Knu98] Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley Professional, 1998.

[KPR+13] Shiva Kintali, Laura J. Poplawski, Rajmohan Rajaraman, Ravi Sundaram, and Shang-Hua Teng. Reducibility among fractional stability problems. *SIAM Journal on Computing*, 42(6):2063–2113, 2013. doi:10.1137/120874655.

[LLD21] Zhengyang Liu, Jiawei Li, and Xiaotie Deng. On the approximation of Nash equilibria in sparse win-lose multi-player games. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*, pages 5557–5565, 2021. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16699.

[LMM03] Richard J. Lipton, Evangelos Markakis, and Aranyak Mehta. Playing large games using simple strategies. In *Proceedings of the 4th ACM Conference on Electronic Commerce (EC)*, pages 36–41, 2003. doi:10.1145/779928.779933.

[Meh18] Ruta Mehta. Constant rank two-player games are PPAD-hard. *SIAM Journal on Computing*, 47(5):1858–1887, 2018. doi:10.1137/15m1032338.

[Pap94] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994. doi:10.1016/S0022-0000(05)80063-7.

[PP21] Christos Papadimitriou and Binghui Peng. Public goods games in directed networks. In *Proceedings of the 22nd ACM Conference on Economics and Computation (EC)*, pages 745–762, 2021. doi:10.1145/3465456.3467616.

[Rub16] Aviad Rubinstein. Settling the complexity of computing approximate two-player Nash equilibria. In *Proceedings of the 57th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 258–265, 2016. doi:10.1109/focs.2016.35.

[Rub18] Aviad Rubinstein. Inapproximability of Nash equilibrium. *SIAM Journal on Computing*, 47(3):917–959, 2018. doi:10.1137/15M1039274.

[Spe28] Emanuel Sperner. Neuer Beweis für die Invarianz der Dimensionszahl und des Gebietes. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 6:265–272, 1928. doi:10.1007/BF02940617.

[SSB17] Steffen Schuldenzucker, Sven Seuken, and Stefano Battiston. Finding clearing payments in financial networks with credit default swaps is PPAD-complete. In *Proceedings of the 8th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 32:1–32:20, 2017. doi:10.4230/LIPIcs.ITCS.2017.32.

[VY11] Vijay V. Vazirani and Mihalis Yannakakis. Market equilibrium under separable, piecewise-linear, concave utilities. *Journal of the ACM*, 58(3):10:1–10:25, 2011. doi:10.1145/1970392.1970394.