Theoretical Computer Science 980 (2023) 114218

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

PPAD-complete approximate pure Nash equilibria in Lipschitz games ☆

Paul W. Goldberg, Matthew Katzman *

Department of Computer Science, University of Oxford, Oxford, England, United Kingdom

ARTICLE INFO

Keywords:
Equilibrium computation
Lipschitz games
Population games
PPAD

ABSTRACT

Lipschitz games, in which there is a limit λ (the Lipschitz value of the game) on how much a player's payoffs may change when some other player deviates, were introduced about 10 years ago by Azrieli and Shmaya. They showed via the probabilistic method that n-player Lipschitz games with m strategies per player have ϵ-approximate *pure* Nash equilibria, for $\epsilon \geq \lambda\sqrt{8n\log(2mn)}$. Here we provide the first hardness result for the corresponding computational problem, showing that even for a simple class of Lipschitz games (Lipschitz polymatrix games), finding ϵ-approximate pure equilibria is PPAD-complete, for suitable pairs of values $\epsilon(n)$, $\lambda(n)$. Novel features of this result include *both* the proof of PPAD hardness (in which we apply a population game reduction from unrestricted polymatrix games) and the proof of containment in PPAD (by derandomizing the selection of a pure equilibrium from a mixed one). In fact, our approach implies containment in PPAD for any class of Lipschitz games where payoffs from mixed-strategy profiles can be deterministically computed. When instead considering games where only payoffs from *action* profiles can be deterministically computed, we provide two equivalent definitions of "randomized PPAD" and show that the generalized problem belongs to this class.

1. Introduction

The basic setting of game theory models a finite game as a finite set of *players*, each of whom chooses from a finite set of allowed *actions* (or *pure strategies*). Such a game maps every *action profile* (a combination of strategies, one for each player) to a *payoff* for each player. It follows that if players are allowed to randomize over their actions, there is a well-defined notion of expected payoff for each player. Nash's famous theorem [30] states that there exist randomized (or "mixed") strategies for the players so that no player can improve their expected payoff by unilaterally deviating to play some alternative strategy. In this paper we make a standard assumption that all payoffs lie in the range $[0, 1]$, and we take an interest in ϵ-approximate Nash equilibria, in which no single player can improve their expected payoff by more than some small additive $\epsilon > 0$ by deviating.

The defining property of a *λ-Lipschitz game* is that, given an action profile, a unilateral deviation by any single player to another action affects every other player's payoff by at most an additive λ. This property of a multiplayer game is of course analogous to the notion of Lipschitz continuity in mathematical analysis. As shown by Azrieli and Shmaya [2], for $\lambda \leq \epsilon/\sqrt{8n\log(2mn)}$, any n-player, m-action, λ-Lipschitz game has an ϵ-approximate Nash equilibrium *in pure strategies*. This is shown by taking a mixed-strategy Nash

☆ This article belongs to Section A: Algorithms, automata, complexity and games, Edited by Paul Spirakis.

* Corresponding author.
E-mail addresses: paul.goldberg@cs.ox.ac.uk (P.W. Goldberg), matthewkatzman26@gmail.com (M. Katzman).

https://doi.org/10.1016/j.tcs.2023.114218
Received 3 January 2023; Received in revised form 30 June 2023; Accepted 20 September 2023

Available online 26 September 2023

equilibrium, known to exist by [30], and showing that when pure strategies are sampled from it, there is a positive probability that it will be ϵ-approximate. As observed in [2], solutions based on *mixed* strategies are often criticized as being unrealistic in practice, and pure strategy solutions are more plausible. However the existence proof of [2] is non-constructive, raising the question of how hard they are to discover.

Here we study the problem of *computing* approximate pure equilibria of Lipschitz games. It is not always sufficient to know that an object exists but not how hard it is to find. Because there is a simple algorithm for verifying approximate Nash equilibria, many versions of this problem belong to TFNP, the class of problems that *always* have a solution that is polynomial-time verifiable. A few subclasses of TFNP have been found to be most relevant when it comes to finding Nash equilibria, each based on the proof technique that can be used to show that every instance of the problem has a solution:

- PPAD [31], the main class we consider in this work, is the class of problems that are proven total via a parity argument on directed graphs.
- The class PLS [26] contains the problems typically associated with polynomial local search and is more frequently the relevant class when considering pure equilibria.
- The class CLS [11] (recently shown equivalent to PPAD $\cap$ PLS) also frequently arises in Nash equilibrium-finding contexts, and is now known to capture the complexity of gradient descent [17].

Significant progress has been made on classifying various subclasses of Lipschitz games (see Section 1.1).

In a Lipschitz *polymatrix* game (the primary subclass we consider in this work), the effect of any one player A's action on the payoffs of any other player B is both bounded and independent of what the remaining players are doing. For each of B's available actions, B's payoff is just the sum of a collection of contributions (each at most λ) associated with the action of each other player, plus some constant shift. For these simple games, we identify values of λ and ϵ (as functions of n) for which this problem is PPAD-complete, and thus unlikely to have a polynomial-time algorithm. Containment in PPAD holds for more general subclasses of Lipschitz games; in essence, any for which one can deterministically and efficiently compute payoffs associated with mixed-strategy profiles. However, for Lipschitz games whose payoff function is computed by a general circuit (or more abstractly, a payoff oracle for action profiles), we instead show containment in BPPPAD - a randomized analogue of PPAD defined in Section 4.

1.1. Background, related work

Due to the guaranteed existence [30] of a mixed Nash equilibrium that can be easily checked (either an exact one, or an ϵ-approximate one for some $\epsilon > 0$), finding such an equilibrium cannot be NP-hard unless NP = co-NP [28]. Beginning with [10,8], the complexity class PPAD (defined below) emerged as the means of identifying whether a class of games is likely to be hard to solve. Until recently, PPAD-completeness has mostly been used to identify computational obstacles to finding mixed equilibria rather than pure ones. There are, however, some examples of Bayesian games, as well as games with an infinite action space, for which PPAD-completeness has been established for finding pure Nash equilibria.

In the setting of Bayes-Nash equilibrium, Filos-Ratsikas et al. [18] obtain a PPAD-completeness result for pure strategy solutions of a class of first-price auctions. Closer to the present paper is a PPAD-completeness result of Papadimitriou and Peng [32] for a class of public goods games with divisible goods, where the continuous action-space of the players can represent probabilities in the solution of a corresponding 2-player game. Here instead we approximate continuous values (those taken at the gates of an arithmetic circuit for which we seek a fixed point) by the actions of a subset of the players, interpreting a given estimate as the fraction of players in such a subset who play some action.

1.1.1. Lipschitz games

Lipschitz games are a well-studied topic formally introduced by Azrieli and Shmaya in [2] as a natural class of games that admit approximate pure Nash equilibria. Such games are often successful at modeling interactions between a large number of agents and arise in many common situations such as financial markets and various types of network games. For example, consider a traffic interaction in which a large number of drivers are attempting to travel from home to work. In a simple enough model, no single driver can have that much impact on any other driver, but collective action can certainly affect the overall congestion [2]. Similarly, in a large economic interaction, any given individual's choice to purchase or not purchase an item may barely affect any other individual's utility, but as the collective demand increases the price is likely to rise, beginning to affect the payoffs of all involved in a continuous manner [20]. In general, the Lipschitz parameter is interpreted as the slope of a player's utility as a (more or less) continuous function of the other players' actions.

These games are assumed to have a large number of players, and introduce the (Lipschitz) restriction that every player can have at most a bounded impact on the payoffs of any other player. They exhibit many fascinating properties, such as various metrics of fault tolerance (explored in [24]) and ex-post stability of equilibria (studied in [13]).

Following their introduction, further work on equilibria of Lipschitz games by Daskalakis and Papadimitriou [12] shows that for *anonymous* Lipschitz games (e.g. the driving example from above, where all other drivers' actions can be permuted and a given player's payoff remains the same, or the economic interaction example where it doesn't matter who makes the purchases so long as they are made), approximate pure equilibria can be computed in polynomial time via reduction to MaxFlow, even for an approximation factor *independent* of the number of players (i.e. $\epsilon = O(\lambda m)$ for λ-Lipschitz, m-action games). When such games are restricted to 2 actions, there is an algorithm that can find such equilibria making polynomially-many *approximate* queries [22] (where

the query may be off by some additive error), and even a randomized algorithm running in polynomial time using only *best-response* queries [3]. Cummings *et al.* [9] exploit a concentration argument to generalize [12] and identify approximate pure equilibria in *aggregative games*.

In [19], Goldberg and Katzman study Lipschitz games from the perspective of *query complexity*, in which an algorithm has black-box access to the game's payoff function. They identify exponential lower bounds on the number of payoff queries needed to find an equilibrium. This leaves open the problem of computing an equilibrium of a Lipschitz game that has a *concise* representation: if a game is known to be concisely representable, then its query complexity is low [21]. Here we show that for concisely-representable Lipschitz games, there remains a computational obstacle. Prior to [19], [20] gave a query-based algorithm for computing $1/8$-approximate Nash equilibria in $1/n$-Lipschitz games; there may be scope to improve on the constant of $1/8$, but the present paper indicates a limit to further progress.

1.1.2. Polymatrix games

Another line of work explores the computational properties of equilibria in *polymatrix* games introduced in [25], in which each pair of players play a different bimatrix game and a player's payoff is the sum of the payoffs from each bimatrix game they play. [7] and [11] show that, for *coordination-only* polymatrix games (games in which each of the bimatrix games always result in the same payoff for both players) finding pure Nash equilibria is PLS-complete while finding mixed Nash equilibria is contained in CLS. Furthermore, [33] and [14] show that the problem of finding ϵ-approximate Nash equilibria of *general* polymatrix games is PPAD-complete for $\epsilon < 0.088$. To complement this lower bound, [16] provide an upper bound in the form of an algorithm running in time polynomial in the number of players to find an ϵ-approximate Nash equilibrium of such games for any constant $\epsilon > 0.5$ *independent of the number of actions*, while in the binary-action case [14] once again improve this result to a polynomial-time algorithm for finding a $\frac{1}{3}$-WSNE (see the definition of a WSNE in Section 2). Polymatrix games are known to be hard to solve even for sparse win-lose matrices [27], and for tree polymatrix games [15].

1.2. Our contributions

In its most general form, the problem we address is, for $\lambda \leq \epsilon/\sqrt{8n\log(2mn)}$:

Given as input an n-player, m-action game, find either an ϵ-approximate pure Nash equilibrium, or a witness that the game is not λ-Lipschitz for the above λ (a witness consists of two action profiles whose payoffs show the game is not λ-Lipschitz).

In particular, we extend the PPAD-hardness results of [14] to games having the Lipschitz property in addition to being polymatrix and binary-action. We show PPAD containment for the problem of finding approximate *pure* Nash equilibria of this class. We will consider m, ϵ, and λ to be parameters of the problem statement rather than provided in the input, however the problem is similar if these are instead passed in as input.

Notice that the two alternative kinds of solutions (an equilibrium *or* a witness against the λ-Lipschitz property) avoid a promise problem: there is no hard-to-check promise that a given game is Lipschitz. The problem statement can be taken to apply in general to any class of games whose payoff functions are efficiently computable (e.g. circuit games [34]). Efficient computability of payoffs is also a sufficient condition to guarantee that solutions are easy to check, allowing us to check the validity of either kind of solution. In the special case of Lipschitz *polymatrix* games of interest here, they can be presented in a straightforward format for which we can check the λ-Lipschitz property.

A PPAD-completeness result consists of containment in PPAD as well as PPAD-hardness. We will rely on the following Lemma taken from the proof technique of [10]:

Lemma 1 *([10]). In the context of* mixed-strategy *Nash equilibrium computation, the following property of a class of games is sufficient for containment in* PPAD*:*

- *Given a mixed-strategy profile, it is possible to deterministically compute the payoffs with additive error $O(\epsilon)$, in time polynomial in the representation size of the game.*
- *$1/\epsilon$ is of size polynomial in the representation size of the game.*

This property is shared by most well-studied classes of multiplayer games, such as graphical games or polymatrix games. It does not hold for unrestricted Lipschitz games,[1] but it of course holds for Lipschitz polymatrix games. In this paper we obtain PPAD containment for approximate *pure* equilibria. The technique of [2] places the problem in a notion of "randomized PPAD", which we rigorously define and explore in greater depth in Section 4. We instead establish containment in PPAD via a *deterministic* method to selecting an approximate pure equilibrium (Lemma 4). Our approach requires λ to be a smaller-growing function of ϵ and n than

[1] For Lipschitz games presented in terms of unrestricted circuits that compute payoffs from action profiles, one cannot deterministically and efficiently compute approximate payoffs to mixed-strategy profiles unless P = RP. For games abstractly presented via a payoff query oracle, this is unconditionally not possible to achieve (regardless of any complexity class collapses).

that of [2], however there *is* overlap between the (ϵ, λ) regime where the problem is contained in (deterministic) PPAD and the one where the problem is hard for PPAD.

PPAD-hardness applies to the binary-action case where each player has two actions $\{1,2\}$. We reduce from the polymatrix games of [14], by considering an induced *population game* (see Section 3.2), which has the Lipschitz property.

For any fixed number of actions $m \geq 2$, PPAD-completeness (Theorem 3 and Corollary 1) holds for carefully-chosen functions $\epsilon(n)$ and $\lambda(n)$. In particular, $\lambda(n)$ needs to be sufficiently small relative to $\epsilon(n)$ in order for our derandomization approach to achieve containment in PPAD (see Section 3.1, Theorem 4). At the same time $\lambda(n)$ needs to be large enough that we still have hardness for PPAD (Section 3.2, Theorem 5). Theorem 3 and Corollary 1 identify $\epsilon(n)$, $\lambda(n)$ for which both of these requirements hold. In particular, taking $\lambda = \Theta(\epsilon n^{-\alpha})$, we present below the current state of the problem for various values of α:

$\alpha < \frac{1}{2}$	No pure equilibrium guarantee [2]
$\alpha \in (\frac{1}{2}, \frac{2}{3})$	PPAD-hard, upper bound unknown (Theorem 5)
$\alpha \in (\frac{2}{3}, 1)$	PPAD-complete (Corollary 1)
$\alpha > 1$	P (trivial to find)

Finally, in Section 4 we initiate our discussion of randomized analogies of PPAD and formalize this concept. We present definitions based on both randomized mapping reductions and on randomized Turing reductions, and show that (in the same way as the deterministic versions [6]) the definitions are equivalent. With the concept well-defined, we prove that the more general problem of finding approximate pure equilibria of Lipschitz games belongs to randomized PPAD via a combination of the techniques of [10] and [2].

2. Preliminaries

We use the following basic notation throughout.

- Boldface letters denote vectors; the symbol $\mathbf{a}$ is used to denote an action profile, and $\mathbf{p}$ is used when the profile may contain mixed strategies.
- $[n]$ and $[m]$ denote the sets $\{1, \ldots, n\}$ of players and $\{1, \ldots, m\}$ of actions, respectively. Furthermore, $i \in [n]$ will always refer to a player, and $j \in [m]$ will always refer to an action.

2.1. Basic game theoretic and complexity concepts

We first introduce standard concepts of strategy profiles, payoffs, regret, and pure/mixed equilibria, followed by the complexity class PPAD. We mostly consider binary-action games (where $m = 2$) with actions labeled 1 and 2 (for more general results considering $m > 2$, see Section 3.1.2).

Types of strategy profile

- An action profile (or, used interchangeably throughout this work, a *pure* strategy profile) $\mathbf{a} = (a_1, \ldots, a_n) \in [m]^n$ is an assignment of one action to each player. We use $\mathbf{a}_{-i} = (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n) \in [m]^{n-1}$ to denote the set of actions played by players in $[n] \setminus \{i\}$ (whenever no player has more than m possible actions, we can assume without loss of generality that every player has exactly m actions as "dummy" actions can always be added that do not affect payoffs or equilibria).
- A (possibly *mixed*) strategy profile $\mathbf{p} = (p_1, \ldots, p_n) \in (\Delta([m]))^n$, where $\Delta([m])$ is the probability simplex over $[m]$ and p_i represents the probability distribution player i assigns over the m actions. When $m = 2$, we abuse the notation p_i to explicitly denote the numerical probability with which player i plays action 2. When $m > 2$, we more generally use $p_{i,j}$ to represent the probability that player i allocates to action j. The set of distributions for players in $[n] \setminus \{i\}$ is denoted $\mathbf{p}_{-i} = (p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n)$. When $\mathbf{p}$ contains just 0-1 values, $\mathbf{p}$ is equivalent to some action profile $\mathbf{a} \in [m]^n$.
- One can consider a tuple of n random variables $\mathbf{a} = (a_1, \ldots, a_n)$, where $a_i = j \in [m]$ with probability $p_{i,j}$. Hence, a random realization of those random variables according to a given strategy profile is an action profile. We denote this by $\mathbf{a} \sim \mathbf{p}$, and similarly, we define $\mathbf{a}_{-i} \sim \mathbf{p}_{-i}$.

Notation for payoffs Given player i, action j, and action profile $\mathbf{a}$,

- $u_i(j, \mathbf{a}_{-i})$ is the payoff that player i obtains for playing action j when all other players play the actions given in $\mathbf{a}_{-i}$.
- $u_i(\mathbf{a}) = u_i(a_i, \mathbf{a}_{-i})$ is the payoff that player i obtains when all players play the actions given in $\mathbf{a}$.
- Similarly for mixed-strategy profiles: $u_i(j, \mathbf{p}_{-i}) = \mathbb{E}_{\mathbf{a}_{-i} \sim \mathbf{p}_{-i}}[u_i(j, \mathbf{a}_{-i})]$ and $u_i(\mathbf{p}) = \mathbb{E}_{\mathbf{a} \sim \mathbf{p}}[u_i(\mathbf{a})]$.

Solution concepts

Definition 1 *(Best response).* Given a player i and a strategy profile $\mathbf{p}$, define the *best response*

$$\mathrm{br}_i(\mathbf{p}) = \arg\max_{j\in[m]} u_i(j, \mathbf{p}_{-i}).$$

In addition, for $\epsilon > 0$, any action j' satisfying

$$u_i(\mathbf{p}) - u_i(j', \mathbf{p}_{-i}) \leq \epsilon$$

is called an ϵ-*best response*.

Note that the best response is defined specifically as an action (or set of actions), rather than potentially mixed strategies, as any action in the support of a mixed strategy that maximizes the utility of player i must also maximize player i's utility.

Definition 2 *(Regret).* Given a player i and a strategy profile $\mathbf{p}$, define the regret

$$\mathrm{reg}_i(\mathbf{p}) = \max_{j\in[m]} u_i(j, \mathbf{p}_{-i}) - u_i(\mathbf{p})$$

to be the difference between the payoffs of player i's best response to $\mathbf{p}_{-i}$ and i's strategy p_i.

Definition 3 *(Equilibria).* We consider the following three types of Nash equilibrium:

- An ϵ-*approximate Nash equilibrium* (ϵ-*ANE*) is a strategy profile $\mathbf{p}^*$ such that, for every player $i \in [n]$, $\mathrm{reg}_i(\mathbf{p}^*) \leq \epsilon$.
- An ϵ-*well supported Nash equilibrium* (ϵ-*WSNE*) is an ϵ-ANE $\mathbf{p}^*$ for which every action j in the support of p_i^* is an ϵ-best response to $\mathbf{p}_{-i}^*$.
- An ϵ-*approximate pure Nash equilibrium* (ϵ-*PNE*) is an action profile $\mathbf{a}$ such that, for every player $i \in [n]$, $\mathrm{reg}_i(\mathbf{a}) \leq \epsilon$.

The complexity class PPAD We assume the reader is familiar with the classes of problems P and NP and the concepts of asymptotic behavior (for an introduction to these concepts, see e.g. [1]). Below we provide the detailed definition of PPAD, introduced in [31].

PPAD is defined by a problem that is complete for the class, the End of Line problem.

Definition 4 *(*End of Line*).* Define End of Line to be the problem taking as input a graph $G = (V, E)$, where the vertex set V is the exponentially-large set $\{0,1\}^n$, and E is encoded by two circuits P and S, where G contains directed edge (u, v) if and only if $S(u) = v$ and $P(v) = u$. Every vertex has both in-degree and out-degree at most one, and $0^n \in V$ has indegree 0, outdegree 1. The problem is to output another vertex v with total degree (i.e. sum of indegree and outdegree) 1.

Definition 5 *(*PPAD*).* PPAD is the set of problems many-one reducible to End of Line in polynomial time. A problem belonging to PPAD is PPAD-complete if End of Line reduces to that problem.

Here we do not use End of Line directly; we reduce from another pre-existing PPAD-complete problem (the general polymatrix mixed Nash equilibrium problem of [14], which in turn reduces from the Pure Circuit problem).

As a final notational comment, throughout the work we use poly(n) wherever any polynomial function in n may be used.

2.2. Lipschitz polymatrix games

In this section we define the classes of games we consider, followed by the associated problem we solve. We will first define binary-action Lipschitz games, before extending the definition to more actions.

Definition 6 *(Lipschitz games [2]).* For any value $\lambda \in (0, 1]$, a binary-action λ-*Lipschitz game* is a game in which a change in action of any given player can affect the payoffs of any other player by at most an additive λ, i.e. for every player i and pair of action profiles $\mathbf{a}, \mathbf{a}'$ with $a_i = a'_i$,

$$|u_i(\mathbf{a}) - u_i(\mathbf{a}')| \leq \lambda ||\mathbf{a}_{-i} - \mathbf{a}'_{-i}||_0$$

(equivalently, since action profiles are 0-1 vectors, the ℓ_1 norm can be used, and is indeed more relevant when it comes to mixed strategies). Note that, under the definition of payoffs to mixed strategies presented above, because mixed strategies of binary-action games are still represented as vectors (of scalars), this can be easily extended to mixed profiles $\mathbf{p}, \mathbf{p}'$ with $p_i = p'_i$:

$$|u_i(\mathbf{p}) - u_i(\mathbf{p}')| \leq \lambda ||\mathbf{p}_{-i} - \mathbf{p}'_{-i}||_1.$$

The above definition, while presented in a different way, is mathematically equivalent to that given by [2] (just provided here in a form more relevant to the current work).

In order to extend the definition above to $m > 2$ actions, we will require a measure of distance for probability distributions:

Definition 7 *(Total variation distance).* The *total variation distance* between distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ over discrete sample space $\mathcal{X}$ (each represented by m-dimensional vectors) is

$$d_{\mathrm{TV}}(\mathcal{D}_1, \mathcal{D}_2) = \frac{1}{2}||\mathcal{D}_1 - \mathcal{D}_2||_1 = \frac{1}{2}\sum_{x \in \mathcal{X}} \left| \Pr_{x' \sim \mathcal{D}_1}(x' = x) - \Pr_{x' \sim \mathcal{D}_2}(x' = x) \right|.$$

Definition 8. For $m > 2$, the pure strategy definition remains the same as in the $m = 2$ case. However, now that mixed strategies are represented by vectors of distributions, the Lipschitz property becomes

$$|u_i(\mathbf{p}) - u_i(\mathbf{p}')| \leq \lambda \sum_{i'=1}^{n} d_{\mathrm{TV}}(p_{i'}, p'_{i'})$$

where d_{TV} is the total variation distance between two distributions (i.e. an extension of the ℓ_1 norm definition above).

We will rely on the total variation distance implicitly throughout the remainder of this work via the following Lemma.

Lemma 2. *Given any player i, the payoff function u_i is λ-Lipschitz with respect to the total variation distance of the mixed strategies of every other player. Furthermore, the regret function* reg_i *is 2λ-Lipschitz in this same measure.*

Proof. The first statement follows directly as a restatement of Definition 8 - when a player i moves probability ρ between actions in a mixed strategy profile, every other player's payoffs may be affected by at most an additive $\lambda\rho$.

Furthermore, a player's best-response payoff function, being the max of a finite number of λ-Lipschitz functions, is itself a λ-Lipschitz function. As such, player i's regret (the difference between player i's payoff and player i's best-response payoff) is 2λ-Lipschitz in the same way. □

With Lipschitz games now defined, we move on to *polymatrix* games, first introduced in [25]. We begin with the simpler concept of *bimatrix* games:

Definition 9 *(Bimatrix games).* An m-action bimatrix game is a game between two players defined by two $m \times m$ matrices of payoffs (one for player 1 and one for player 2) for each action-action pair.

With these defined, we can extend the definition to polymatrix games:

Definition 10 *(Polymatrix games).* A degree-D, n-player, m-action polymatrix game G is an undirected graph (V, E) of degree D in which each vertex represents a player and each edge $(i_1, i_2) \in E$ a bimatrix game between players i_1 and i_2. There is a set of payoffs β_{i_1,i_2,j_1,j_2} for every such bimatrix game and every pair of actions $j_1, j_2 \in [m]$ such that each player's payoff in G is the sum of their payoffs from each of the (at most D) bimatrix games they play.

Note that, as defined, such a game is λ-Lipschitz for any

$$\lambda \geq \max_{i,i' \in [n], j, j'_1, j'_2 \in [m]} \left| \beta_{i,i',j,j'_1} - \beta_{i,i',j,j'_2} \right|$$

(i.e. the maximum payoff change any player experiences from any other player's change of action).

In order to analyze these games, it is helpful to understand the size required to represent them. Given a payoff oracle, it is easy to verify that a polymatrix game is λ-Lipschitz just by querying each of the nDm^2 payoffs $\beta_{i,i',j,j'}$ and ensuring the above inequality holds. $\beta_{i,i',j,j'}$ represents a small contribution, possibly negative, that i' makes to i (when i plays j) by playing j'. To be valid, all total payoffs must lie in $[0, 1]$, which can be checked from a game presented via the quantities $\beta_{i,i',j,j'}$ by performing the following additional checks for every $i \in [n], j \in [m]$:

$$\sum_{i' \neq i} \max_{j' \in [m]} \beta_{i,i',j,j'} \leq 1$$

$$\sum_{i' \neq i} \min_{j' \in [m]} \beta_{i,i',j,j'} \geq 0$$

There are $2nm$ such calculations, each of which requires $O(nm)$ operations, so an invalid game can be uncovered in polynomial time.

Remark 1. Lipschitz polymatrix games are a strict subset of Lipschitz games and require only $O(n^2m^2)$ queries to learn completely and can thus be concisely represented in $O(n^2m^2)$ space (by providing the value of each β_{i_1,i_2,j_1,j_2}).

We now define the main problem we consider in this work.

Definition 11. Define (m, ϵ, λ)-PURELPG to be the problem of finding ϵ-PNEs of n-player, m-action, λ-Lipschitz polymatrix games, or alternatively finding a witness that the game is not λ-Lipschitz (note once more that m, ϵ, and λ are parameters of the problem while n is provided as input). Similarly, define (m, ϵ, λ)-MIXEDLPG to be the corresponding problem for mixed equilibria.

Remark 2. Note that, while we will often fix the number of actions m, both ϵ and λ are often functions of the number of players n. We generally think of these as being decreasing functions of n.

We are interested in the complexity of the problem, for various pairs of functions $\epsilon(n)$, $\lambda(n)$.

Remark 3. One basic observation we make is that, for any $a \in (0, 1)$, (m, ϵ, λ)-PURELPG reduces to $(m, a\epsilon, a\lambda)$-PURELPG, by rescaling payoffs.

We extend the following results of [33] and, more recently, of [14] to pure equilibria in *Lipschitz* polymatrix games:

Theorem 1 *([33]). There exists some constant $\epsilon > 0$ such that given a binary-action polymatrix game, finding an ϵ-ANE is* PPAD*-complete.*

Theorem 2 *([14]). It is* PPAD*-complete to find a 0.088-ANE of an n-player, m-action, degree-D polymatrix game, even when $m = 2$ and $D = 3$.*

3. Results

The result we achieve is as follows:

Theorem 3 *(Main result). For every constant $m \geq 2$, there exists some constant $\epsilon > 0$ such that, for all functions $\lambda(n) = \Theta(n^{-3/4})$, (m, ϵ, λ)-PURELPG is PPAD-complete.*

In fact, for any constant $\alpha \in (\frac{2}{3}, 1)$, Theorem 3 holds for $\lambda(n) = \Theta(n^{-\alpha})$. In order to prove Theorem 3, we will need to show both containment in PPAD and PPAD-hardness under these ranges of the parameters.

3.1. Containment in PPAD

Theorem 4. *For all functions $\epsilon(n), \lambda(n)$ satisfying*

$$\lambda(n) = \frac{1}{\text{poly}(n)}, \qquad \epsilon(n) = \omega(\lambda(n) n^{2/3})$$

(m, ϵ, λ)-PURELPG $\in$ PPAD.

In particular, we prove this when $\epsilon(n) \geq 6\lambda(n)\sqrt[3]{nDm\log(3m)}$, where $D < n$ is the degree of the game being considered.

3.1.1. Warmup

We will begin with the proof for $m = 2$. The proof of this theorem is broken down further into two smaller steps. First, we will exhibit sufficient settings of the parameters such that the problem of computing a *mixed* approximate Nash equilibrium of a Lipschitz polymatrix game is contained in PPAD. Second, with the parameters set as in the statement of Theorem 4, we describe a derandomization technique for deriving an approximate *pure* equilibrium from the mixed one in polynomial time. More formally:

Lemma 3. *Whenever $\lambda(n) = \frac{1}{\text{poly}(n)}$, $(2, \frac{\lambda}{8}, \lambda)$-MIXEDLPG $\in$ PPAD.*

Proof. This follows from Lemma 1. We require a deterministic computation of mixed payoffs with additive error $O(\epsilon)$ in time polynomial in n and $\frac{1}{\epsilon}$. Since ϵ is an inverse polynomial in n (by the inverse polynomial assumption placed on λ), the key requirement is the ability to answer mixed payoff queries in time polynomial in n. Because the payoffs to a player i in polymatrix games are linear in the actions of every other player, any mixed payoff query can be answered in time $O(n^2 m^2)$ (or simply $O(n^2)$ since we are keeping the number of actions fixed). Thus $(2, \frac{\lambda}{8}, \lambda(n))$-MIXEDLPG $\in$ PPAD. □

Lemma 4 *(Main technical lemma). For functions $\epsilon(n), \lambda(n)$ satisfying $\epsilon(n) \geq \lambda\sqrt[3]{70nD}$, an ϵ-PNE of an n-player, binary-action, λ-Lipschitz, degree-D polymatrix game G can be derived from an $\frac{\lambda}{8}$-ANE of G in polynomial time.*

Proof. Throughout this proof we implicitly use the assumption that payoffs from mixed strategy profiles can be computed in polynomial time (as discussed in the proof of Lemma 1). We will consider the (signed) discrepancy

$$d_i(\mathbf{p}) = u_i(2, \mathbf{p}_{-i}) - u_i(1, \mathbf{p}_{-i})$$

(d_i is a 2λ-Lipschitz function via the same argument as Lemma 2).

Take some $\frac{\lambda}{8}$-ANE $\mathbf{p}^*$ of G. The reduction is performed in three steps:

(1) Constructing from $\mathbf{p}^*$ a well-supported Nash equilibrium $\mathbf{p}^{(0)}$. This step is required because we will use a player's discrepancy as a linear proxy for regret, and general approximate equilibria may allow players to allocate small probabilities to bad actions, thus decoupling the two measures (in contrast with exact equilibria, in which any mixed strategy by definition implies no discrepancy between the two actions). Well-supported equilibria do not suffer from this same decoupling.
(2) Iteratively converting $\mathbf{p}^{(0)}$ into an action profile $\mathbf{a}$ that is "close" to an equilibrium. To achieve this step, we will bound the rate at which the sum of all players' discrepancies is allowed to increase, ensuring that few players experience large regret.
(3) Correcting $\mathbf{a}$ to obtain an approximate pure equilibrium $\mathbf{a}^*$. Because of the guarantees from the previous step, we can achieve this by allowing every player with high regret to change to their best response.

Step 1: Mixed equilibrium to WSNE In this first step we construct WSNE $\mathbf{p}^{(0)}$ from $\mathbf{p}^*$. The idea here is to ensure that all players fall into one of the following two categories:

- Players with discrepancy that is small in absolute value
- Players playing pure strategies

In order to do this, consider any player i such that $|d_i(\mathbf{p}^*)| > \frac{\lambda}{2}\sqrt{D}$; without loss of generality we can assume the discrepancy is negative, i.e. action 1 is significantly better. Then, using the fact that when action 1 is the best response

$$\begin{aligned}\mathrm{reg}_i(\mathbf{p}^*) &= u_i(1, \mathbf{p}^*_{-i}) - \left(p_i^* u_i(2, \mathbf{p}^*_{-i}) + (1 - p_i^*) u_i(1, \mathbf{p}^*_{-i})\right) \\ &= p_i^* u_i(1, \mathbf{p}^*_{-i}) - p_i^* u_i(2, \mathbf{p}^*_{-i}) \\ &= -p_i^* d_i(\mathbf{p}^*)\end{aligned}$$

we obtain

$$\mathrm{reg}_i(\mathbf{p}^*) = -p_i^* d_i(\mathbf{p}^*) > p_i^* \frac{\lambda}{2}\sqrt{D}$$

so, since $\mathbf{p}^*$ is a $\frac{\lambda}{8}$-approximate equilibrium (and therefore $\mathrm{reg}_i(\mathbf{p}^*) \le \frac{\lambda}{8}$),

$$p_i^* < \frac{1}{4\sqrt{D}}.$$

To construct $\mathbf{p}^{(0)}$, every such player (at most n) should change to playing their pure best response. In this case, every player's discrepancy will increase by at most $2D\lambda p_i^*$ (recall that discrepancy is 2λ-Lipschitz), so in this new profile $\mathbf{p}^{(0)}$, no player playing a mixed strategy can experience discrepancy greater than

$$\frac{\lambda}{2}\sqrt{D} + 2D\lambda\frac{1}{4\sqrt{D}} = \lambda\sqrt{D}$$

while no player playing a pure strategy can experience regret greater than

$$0 + 2D\lambda\frac{1}{4\sqrt{D}} = \frac{\lambda}{2}\sqrt{D}.$$

Thus $\mathbf{p}^{(0)}$ is a $\lambda\sqrt{D}$-WSNE, and for any player i playing a mixed strategy, $d_i(\mathbf{p}^*) \le \lambda\sqrt{D}$. We call this $\lambda\sqrt{D}$-WSNE $\mathbf{p}^{(0)}$.

Step 2: WSNE to action profile In this step, we iteratively define intermediate strategy profiles $\mathbf{p}^{(i)}$ and intermediate sets of players $S^{(i)}$ who have, at any point, experienced small discrepancy (defined below to ensure that players not in $S^{(i)}$ cannot have high regret in profile $\mathbf{p}^{(i)}$). Informally, for every player i, $\mathbf{p}^{(i)}$ is a strategy profile in which players $1, \dots, i$ are playing pure strategies and players $i+1, \dots, n$ may not be. Furthermore, the set $S^{(i)}$ is the set of players who, in at least one of the profiles $\mathbf{p}^{(0)}, \dots, \mathbf{p}^{(i)}$ have experienced discrepancy at most $\lambda\sqrt{D}$. These will be iteratively computed to ultimately arrive at $\mathbf{a} = \mathbf{p}^{(n)}$: an action profile that is in some sense "close" to an approximate pure equilibrium.

Counterintutively, this will not always involve assigning each player to play their best response. Instead, we will instruct each player to play the action that allows us to bound the effect on the other players. Roughly, we identify below a measure of badness C that is quadratic in the probability p assigned by a player to action 2 as opposed to action 1. In setting p to 0 or 1, we minimize the linear term in this measure, choosing 0 or 1 depending on the sign of the derivative with respect to p, and the measure increases by the relatively small second-order term.

More formally, take $\mathbf{p}^{(0)}$ as above and define

$$S^{(0)} = \{i' : |d_{i'}(\mathbf{p}^{(0)})| \leq \lambda\sqrt{D}\}$$

(i.e. a superset of the players playing mixed strategies in $\mathbf{p}^{(0)}$) while

$$S^{(i)} = S^{(i-1)} \cup \{i' : |d_{i'}(\mathbf{p}^{(i)})| \leq \lambda\sqrt{D}\}$$

(adding at each step the set of players playing pure strategies in $\mathbf{p}^{(0)}$ but experiencing small discrepancy for the first time at step i; note that, due to the Lipschitz property, no player's discrepancy can change sign without that player's discrepancy first falling below $\lambda\sqrt{D}$ and that player joining $S^{(i)}$, so any player not in $S^{(i)}$ must be playing their pure best response). Furthermore, define the sum of squared discrepancies cost function (which considers only the players in $S^{(i)}$):

$$C(\mathbf{p}^{(i)}) = \sum_{i' \in S^{(i)}} d_{i'}^2(\mathbf{p}^{(i)}).$$

Note that C only considers players in the set $S^{(i)}$, as it is meant to be a proxy for how far we are from equilibrium, and players in $[n] \setminus S^{(i)}$ have high discrepancies but are playing their best responses and hence should not contribute to this distance to equilibrium.

It is difficult to minimize C itself, and regardless C is already only a proxy for the true objective we must minimize to achieve an approximate pure equilibrium. However, since $d_{i'}$ is a multi-linear function (in fact, in the case of polymatrix games, linear), we can write

$$d_{i'}(\mathbf{p}^{(i)}) = c + \ell p_i$$

where c is some constant dependent only on $\mathbf{p}_{-i}^{(i)}$ and ℓ is a coefficient with absolute value at most 2λ (the Lipschitz parameter of $d_{i'}$). Squaring this, and keeping in mind that $\mathbf{p}_{-i}^{(i-1)} = \mathbf{p}_{-i}^{(i)}$, we can expand (for every $i' \in S^{(i-1)}$):

$$d_{i'}^2(\mathbf{p}^{(i)}) - d_{i'}^2(\mathbf{p}^{(i-1)}) = \alpha(p_i^{(i)} - p_i^{(i-1)}) + \ell^2((p_i^{(i)})^2 - (p_i^{(i-1)})^2)$$

for some value $\alpha_i = 2c\ell$ dependent only on $\mathbf{p}_{-i}^{(i)}$. Taking the sum over all players $i' \in S^{(i)}$, this yields

$$C(\mathbf{p}^{(i)}) - C(\mathbf{p}^{(i-1)}) = A(p_i^{(i)} - p_i^{(i-1)}) + \Lambda((p_i^{(i)})^2 - (p_i^{(i-1)})^2) + K_i$$

where

- A may be negative or non-negative.
- $\Lambda \leq 4\lambda^2 D$ (and obviously $(p_i^{(i)})^2 - (p_i^{(i-1)})^2 \leq 1$).
- K_i represents the one-time contribution of at most $\lambda^2 D$ each from any player in $S^{(i)} \setminus S^{(i-1)}$ whose discrepancy will be counted in this and all future costs. While we can't say anything about each individual K_i (since we don't know when exactly every player's discrepancy will dip to this level), we note that there are at most n such contributions of $\lambda^2 D$ (one for every time a player joins the set of small-discrepancy players), i.e.:

  $$\sum_{i \in [n]} K_i \leq \lambda^2 nD$$

  (this bound is the reason we needed to start from a WSNE).

If $p_i^{(i-1)} \in \{0, 1\}$, then set $\mathbf{p}^{(i)} = \mathbf{p}^{(i-1)}$ (if a player is already playing a pure strategy, we can avoid adding to C at all on their turn). Otherwise, since we can efficiently compute A and decide whether it is negative or not, we can select $p_i^{(i)}$ such that $A(p_i^{(i)} - p_i^{(i-1)}) \leq 0$ (one such option for $p_i^{(i)} \in \{0, 1\}$ must exist). This bounds

$$C(\mathbf{p}^{(i)}) - C(\mathbf{p}^{(i-1)}) \leq 4\lambda^2 D + K_i$$

and a quick induction achieves

$$C(\mathbf{p}^{(n)}) \leq 4\lambda^2 nD + \sum_{i \in [n]} K_i \leq 4\lambda^2 nD + \lambda^2 nD = 5\lambda^2 nD.$$

Furthermore, $\mathbf{a} = \mathbf{p}^{(n)}$ is an action profile.

Step 3: Action profile to pure equilibrium Here we distinguish between players in $S^{(n)}$ and players not in $S^{(n)}$.

The players not in $S^{(n)}$ must be playing their best response in $\mathbf{a}$. This is because they were playing their best response in $\mathbf{p}^{(0)}$, and their discrepancy never approached 0 or changed sign, so their best response must never have changed. Therefore, none of these players suffer any regret.

We now consider the players in $S^{(n)}$. Note that we have bounded the sum of the squares of the discrepancies of the players in this set to be at most $5\lambda^2 nD$. We can now also bound the number of players with regrets that are too high and fix them. To begin this final step, define $\delta = \lambda\sqrt[3]{20nD}$ and consider the number of players $i \in S^{(n)}$ such that $d_i(\mathbf{a}) > \delta$. This will be at most $\frac{C(\mathbf{a})}{\delta^2}$, since $C(\mathbf{a})$

is the sum of squared discrepancies. So have every player with *regret* at least δ in profile $\mathbf{a}$ (a subset of those with high *discrepancy*) simultaneously switch actions. Then, once more invoking Lemma 2, the maximum regret any player (in $S^{(n)}$ or otherwise) can experience is

$$\delta + 2\lambda\frac{C(\mathbf{a})}{\delta^2} \leq \delta + \frac{10\lambda^3 nD}{\delta^2} < \lambda\sqrt[3]{70nD}.$$

In other words, the action profile $\mathbf{a}^*$ obtained after these players switch actions is a $\lambda\sqrt[3]{70nD}$-PNE. This completes the proof of Lemma 4. □

Finally, because a violation of the Lipschitz property can be uncovered in a quick check before commencing the above algorithm simply by ensuring all the coefficients are smaller than λ, and [6] shows that PPAD is closed under Turing reductions, the binary-action version of Theorem 4 clearly follows from combining the two Lemmas above.

3.1.2. *Containment in* PPAD *for additional actions*

We will now modify the proof of Theorem 4 to values of $m > 2$. Once again, the proof of this theorem is broken down further into two smaller steps:

Lemma 5 *(Extension of Lemma 3). For any fixed* $m \geq 2, \lambda(n) = \frac{1}{\text{poly}(n)}$,
$(m, \left(\frac{m-1}{m}\right)^2\lambda, \lambda)$-MIXEDLPG $\in$ PPAD.

The proof of Lemma 5 is essentially the same as that of Lemma 3.

Lemma 6 *(Extension of Lemma 4). For functions* $\epsilon(n), \lambda(n)$ *satisfying*

$$\epsilon(n) \geq 6\lambda(n)\sqrt[3]{n^2 m\log(3m)}$$

an $\epsilon(n)$*-PNE of an* n*-player,* m*-action,* λ*-Lipschitz polymatrix game* G *can be derived from an* $\left(\frac{m-1}{m}\right)^2\lambda$*-ANE of* G *in polynomial time.*

Remark 4. Unlike the $m = 2$ case, setting $D < n$ does not improve the quality of the equilibrium we can obtain, so we will assume a general degree-$(n-1)$ polymatrix game.

Proof. We will use the following values in this proof. Let

$$\epsilon_0 = \left(\frac{m-1}{m}\right)^2\lambda$$
$$\epsilon_1 = 2\sqrt{2n\lambda\epsilon_0}.$$

Then, in order to prove Lemma 6, we need to convert an ϵ_0-ANE to an ϵ-PNE. We will do so in the same three steps as in the proof of Lemma 4, but more care must be taken when generalizing the proof.

The primary generalization required is the transition from considering discrepancy to considering variance. Whereas in Lemma 4 there was a well-defined notion of discrepancy that could be expressed as quadratic in the probabilities assigned to the actions of a given player, such a concept does not easily translate to games with more actions. In order to rely on similar techniques, we will instead consider the variance of a subset of each player's actions. The exact subset to be considered will be described below, and throughout this proof will be referred to as the "relevant" set, as it will contain the set of actions that may become relevant in our analysis of the algorithm.

We will proceed along the same lines as the proof of Lemma 4 with the modified approach.

Step 1: Mixed equilibrium to WSNE We begin with an ϵ_0-ANE $\mathbf{p}^*$. Consider any player i and assume that player i puts probability p on actions with regret greater than

$$\delta_0 = \sqrt{2(n-1)\lambda\epsilon_0}.$$

The regret of this player is at least $p\delta_0$, so it must be the case that $p \leq \epsilon_0/\delta_0$.

This being the case, if every player simultaneously moves the probability from such actions to their best response, then any given player's worst-case regret becomes

$$\delta_0 + 2(n-1)\lambda p \leq \delta_0 + \frac{2(n-1)\lambda\epsilon_0}{\delta_0} \leq 2\sqrt{2n\lambda\epsilon_0} = \epsilon_1.$$

Thus an ϵ_0-ANE can be converted easily to an ϵ_1-WSNE.

Step 2: WSNE to action profile This is, once again, the most involved step. Unlike Step 1 (which occurred simultaneously) and Step 3 (which will do the same), this step will take place for each player in turn. We will iteratively define intermediate strategy

profiles $\mathbf{p}^{(i')}$ in which players $1,\ldots,i'$ are playing pure strategies and players $i'+1,\ldots,n$ may not be. Ultimately, we will arrive at $\mathbf{a}=\mathbf{p}^{(n)}$: an action profile that is in some sense "close" to an approximate pure equilibrium.

Counterintuitively, this will not always involve instructing each player to play their best response when their turn arises. Instead, we will instruct each player to play the action that allows us to bound its effect on the other players. Roughly, we identify a measure of badness that is quadratic in the probability assigned by a player to each action. In selecting an action, we minimize the linear term in this measure depending on the direction of the gradient, and the measure increases by the relatively small second-order term. The quadratic measure we select will be the sum of the variances of subsets of the players' actions.

More specifically, we define the following concepts:

Notation	Definition
$S_i^{(i')}$	Player i's **relevant set** of actions (defined below) after step i'
$m_i^{(i')}$	$\left\|S_i^{(i')}\right\|$, i.e. the number of relevant actions player i has after step i'
$\mathbf{u}_i(S,t)$	the length-$\|S\|$ restriction of player i's payoff vector after step t to the actions in set S
$\mu_i(S,t)$	the average value of $\mathbf{u}_i(S,t)$, i.e. the payoff to player i after step t for playing the uniform mixed strategy over all actions in player set S
$\sigma_i^2(S,t)$	the variance of the values in $\mathbf{u}_i(S,t)$
$\mathrm{reg}_i(j,t)$	the regret player i experiences for playing action j after step t

Now, we will define the relevant set via the following recurrence relation. The base case will be:

$$S_i^{(0)}=\{j \mid \mathrm{reg}_i(j,0)\le \epsilon_1\}.$$

The recurrence is:

Listing 1.1. Relevant set recurrence.

```
S_i^(i'+1) = S_i^(i')
μ = μ_i(S_i^(i'+1), i'+1)
while (max_{j∉S_i^(i'+1)} u_i(j, i'+1) ≥ μ):
    S_i^(i'+1) = S_i^(i'+1) ∪ {arg max_{j∉S_i^(i'+1)} u_i(j, i'+1)}
    μ = μ_i(S_i^(i'+1), i'+1)
```

In other words, $S_i^{(i'+1)}$ is the superset of $S_i^{(i')}$ obtained by repeatedly adding player i's highest-paying action to $S_i^{(i'+1)}$ and recalculating the value of $\mu_i(S_i^{(i'+1)},i'+1)$ until doing so would decrease its value. We call $S_i^{(i')}$ the relevant set because it consists of the actions that, at any point during the algorithm's run, are worth consideration in our attempt to bound the total regret experienced by all players.

Step 2 of the algorithm will thus proceed as follows. During step i', player i' will first change their strategy to the action that has the least impact on the linear term of the sum of the variances of all other players' relevant sets. *Then* every player will recalculate their relevant sets using the new payoffs considering the changed action of player i'.

There are three values we need to bound here:

1. The sum of the variances of $S_i^{(0)}$, the players' initial relevant sets:

$$\sum_{i=1}^{n}\sigma_i^2\left(S_i^{(0)},0\right).$$

2. How much larger the sum of the variances of $S_i^{(i'+1)}$ can be than the sum of the variances of $S_i^{(i')}$ (between steps), adding to a total over all n steps of:

$$\sum_{i=1}^{n}\sum_{i'=1}^{n}\sigma_i^2\left(S_i^{(i')},i'\right)-\sigma_i^2\left(S_i^{(i'-1)},i'\right).$$

3. How much the sum of the variances of $S_i^{(i')}$, can increase during step i', adding to a total over all n steps of:

$$\sum_{i=1}^{n}\sum_{i'=1}^{n}\sigma_i^2\left(S_i^{(i'-1)},i'\right)-\sigma_i^2\left(S_i^{(i'-1)},i'-1\right).$$

Note that the sum of these three values is

$$\sum_{i=1}^{n}\sigma_i^2\left(S_i^{(n)},n\right)$$

i.e. the sum of the variances of the relevant sets at the end of the entire Step 2.

We begin with quantity (1). The variance of any set of points within the interval $[a,b]$ is at most $(b-a)^2/4$ (see, for example, [5]), so quantity (1) is at most

$$n\frac{\epsilon_1^2}{4}=2n^2\lambda\epsilon_0=2\left(\frac{n\lambda(m-1)}{m}\right)^2$$

(note that this n^2 cannot be reduced to nD when the degree of the polymatrix game is reduced, limiting the savings we can get by restricting the degree). Next, we bound quantity (2). Quantity (2) considers no change in the strategy profile - all payoffs considered are based on strategy $\mathbf{p}^{(i')}$. Then, for every player, actions are added to the set $S_i^{(i')}$ one by one (each time increasing the average value of the payoffs) to form the set $S_i^{(i'+1)}$. The critical observation is that, in addition to providing a lower bound (the current average) on the payoff of this newly added action, we can also ensure an upper bound of $\mu_i\left(S_i^{(i')},i'\right)+2\lambda$, as the action must not have been relevant in the previous step. So specifically, consider a set of k points $\{x_1,\ldots,x_k\}$ with mean μ and variance σ^2 and consider adding a point x_{k+1} that is within an additive 2λ of μ. We calculate the change in variance below, keeping in mind that if we subtract μ from every point the variance remains the same:

$$\begin{aligned}
&\mathrm{Var}(x_1,\ldots,x_{k+1})-\sigma^2\\
&=\mathrm{Var}(x_1-\mu,\ldots,x_{k+1}-\mu)-\sigma^2\\
&=\left(\frac{1}{k+1}\sum_{j=1}^{k+1}(x_j-\mu)^2\right)-\left(\frac{1}{k+1}\left(\sum_{j=1}^{k}(x_j-\mu)\right)^2\right)-\sigma^2\\
&=\left(\frac{1}{k+1}\left((x_{k+1}-\mu)^2+\sum_{j=1}^{k}(x_j-\mu)^2\right)\right)\\
&\quad-\left(\frac{1}{k+1}\left((x_{k+1}-\mu)+\sum_{j=1}^{k}(x_j-\mu)\right)^2\right)-\sigma^2\\
&=\frac{(x_{k+1}-\mu)^2}{k+1}+\frac{k}{k+1}\sigma^2-\frac{(x_{k+1}-\mu)^2}{(k+1)^2}-\sigma^2\\
&=\frac{1}{k+1}\left(\frac{k}{k+1}(x_{k+1}-\mu)^2-\sigma^2\right)
\end{aligned}$$

where we can go from line 4 to line 5 using the fact that

$$\sum_{j=1}^{k}(x_j-\mu)=0.$$

So, since $|x_{k+1}-\mu|\le 2\lambda$, the largest possible increase in variance is

$$\frac{4k\lambda^2}{(k+1)^2}$$

(and, in fact, adding a point will usually *decrease* the variance, but there's no need to get that specific because this bounded increase will serve our purpose for constant values of m). If we add this contribution for the maximum possible $m-1$ actions (the relevant set will never start empty) and all n players, we see

$$\sum_{i=1}^{n}\sum_{k=1}^{m-1}\frac{4k\lambda^2}{(k+1)^2}\le n\sum_{k=1}^{m-1}\frac{4\lambda^2}{k}=4n\lambda^2H_{m-1}<4n\lambda^2(\log(m-1)+1).$$

Finally, we need to bound quantity (3). Note that, given $\mathbf{u}_i\left(S_i^{(i'-1)},i'-1\right)$, the quantity $\mathbf{u}_i\left(S_i^{(i'-1)},i'\right)$ and its mean and variance are all functions of the strategy vector $\mathbf{p}_{i'}$ of player i'. In particular, we can write:

$$\mathbf{u}_i(S_i^{(i'-1)},i')-\mathbf{1}\mu_i(S_i^{(i'-1)},i')=\mathbf{c}_i^{(i')}+\mathbf{L}_i^{(i')}\mathbf{p}_{i'}^{(i')}$$

for some constant $m_i^{(i'-1)} \times 1$ vector $\mathbf{c}_i^{(i')}$ (which may depend on the known values $\mathbf{u}_i\left(S_i^{(i'-1)}, i'-1\right)$) and $m_i^{(i'-1)} \times m$ coefficient matrix $\mathbf{L}_i^{(i')}$ in which each element has magnitude no more than $\frac{m-1}{m}\lambda$. Propagating this expression:

$$\begin{aligned}\sigma_i^2(S_i^{(i'-1)}, i') &= \frac{1}{m_i^{(i'-1)}} \left|\left|\mathbf{u}_i(S_i^{(i'-1)}, i') - \mathbf{1}\mu_i(S_i^{(i'-1)}, i')\right|\right|_2^2 \\ &= \frac{1}{m_i^{(i'-1)}} \left(\left|\left|\mathbf{c}_i^{(i')}\right|\right|_2^2 + \left|\left|\mathbf{L}_i^{(i')}\mathbf{p}_{i'}^{(i')}\right|\right|_2^2 + 2\mathbf{c}_i^{(i')} \cdot \mathbf{L}_i^{(i')}\mathbf{p}_{i'}^{(i')}\right). \end{aligned} \tag{1}$$

If we specifically consider the mixed term:

$$\begin{aligned}2\mathbf{c}_i^{(i')} \cdot \mathbf{L}_i^{(i')}\mathbf{p}_{i'}^{(i')} &= 2\mathbf{c}_i^{(i)'\intercal}\left(\mathbf{L}_i^{(i')}\mathbf{p}_{i'}^{(i')}\right) \\ &= 2\left(\mathbf{c}_i^{(i')\intercal}\mathbf{L}_i^{(i')}\right)\mathbf{p}_{i'}^{(i')} \\ &= \mathbf{b}_i^{(i')\intercal}\mathbf{p}_{i'}^{(i')} \end{aligned} \tag{2}$$

for some vector $\mathbf{b}_i^{(i')} \in \mathbb{R}^m$. So now we can consider the quantity we're really after. We focus on (for some $i' \in [n]$) the sum over all players $i \in [n]$ of the changes in $\sigma_i^2\left(S_i^{(i'-1)}, \cdot\right)$ after step i':

$$\sum_{i=1}^{n} \sigma_i^2\left(S_i^{(i'-1)}, i'\right) - \sigma_i^2\left(S_i^{(i'-1)}, i'-1\right). \tag{3}$$

We want to minimize this value as best we can, as our goal is to end up with small variance among the set of relevant actions. While the variance is not a very good proxy for the regret (see Equation (5) below), it is a value that we will be able to successfully bound. So, substituting Equations (1) and (2) into Equation (3), we see that the change in this value across step i' is

$$\begin{aligned}&\sum_{i=1}^{n} \frac{1}{m_i^{(i'-1)}} \left(\left|\left|\mathbf{c}_i^{(i')}\right|\right|_2^2 + \left|\left|\mathbf{L}_i^{(i')}\mathbf{p}_{i'}^{(i')}\right|\right|_2^2 + \mathbf{b}_i^{(i')\intercal}\mathbf{p}_{i'}^{(i')} - \left|\left|\mathbf{c}_i^{(i')}\right|\right|_2^2 - \left|\left|\mathbf{L}_i^{(i')}\mathbf{p}_{i'}^{(i'-1)}\right|\right|_2^2 - \mathbf{b}_i^{(i')\intercal}\mathbf{p}_{i'}^{(i'-1)}\right) \\ &= \sum_{i=1}^{n} \frac{1}{m_i^{(i'-1)}} \left(\left|\left|\mathbf{L}_i^{(i')}\mathbf{p}_{i'}^{(i')}\right|\right|_2^2 + \mathbf{b}_i^{(i')\intercal}\mathbf{p}_{i'}^{(i')} - \left|\left|\mathbf{L}_i^{(i')}\mathbf{p}_{i'}^{(i'-1)}\right|\right|_2^2 - \mathbf{b}_i^{(i')\intercal}\mathbf{p}_{i'}^{(i'-1)}\right) \\ &= \sum_{i=1}^{n} \frac{1}{m_i^{(i'-1)}} \left(\left|\left|\mathbf{L}_i^{(i')}\mathbf{p}_{i'}^{(i')}\right|\right|_2^2 - \left|\left|\mathbf{L}_i^{(i')}\mathbf{p}_{i'}^{(i'-1)}\right|\right|_2^2 + \mathbf{b}_i^{(i')\intercal}\left(\mathbf{p}_{i'}^{(i')} - \mathbf{p}_{i'}^{(i'-1)}\right)\right) \\ &\leq \sum_{i=1}^{n} \frac{1}{m_i^{(i'-1)}} \left(\left(\frac{m-1}{m}\lambda\right)^2 + \mathbf{b}_i^{(i')\intercal}\left(\mathbf{p}_{i'}^{(i')} - \mathbf{p}_{i'}^{(i'-1)}\right)\right) \\ &= \mathbf{b}^{(i')\intercal}\left(\mathbf{p}_{i'}^{(i')} - \mathbf{p}_{i'}^{(i'-1)}\right) + \sum_{i=1}^{n} \frac{1}{m_i^{(i'-1)}} \left(\frac{m-1}{m}\lambda\right)^2 \end{aligned} \tag{4}$$

for some vector $\mathbf{b}^{(i')} \in \mathbb{R}^m$ (the sum of the vectors $\frac{1}{m_i^{(i'-1)}}\mathbf{b}_i^{(i')}$). So, ultimately, our algorithm aims to minimize this upper bound by ensuring that the first term is not positive. Recall that $\mathbf{p}_{i'}^{(i'-1)}$ is a (given) vector of non-negative real numbers with $\left|\left|\mathbf{p}_{i'}^{(i'-1)}\right|\right|_1 = 1$, while $\mathbf{p}_{i'}^{(i')}$ is a (desired) vector containing $m-1$ zeros and a single one. Consider the element b (at index j) of $\mathbf{b}^{(i')}$ with the smallest value *among those indices in the relevant set* $S_{i'}^{(i')}$ (note that player i''s entire support will be contained in $S_{i'}^{(i')}$). Set $\mathbf{b} = \mathbf{e}_j$ (the standard basis vector in dimension j). This ensures that the first term in Equation (4) is not positive.

So, proceeding iteratively as described, this ensures

$$\sum_{i=1}^{n}\sum_{i'=1}^{n} \sigma_i^2\left(S_i^{(i'-1)}, i'\right) - \sigma_i^2\left(S_i^{(i'-1)}, i'-1\right) \leq \sum_{i=1}^{n}\sum_{i'=1}^{n} \frac{1}{m_i^{(i'-1)}}\left(\frac{m-1}{m}\lambda\right)^2 \leq \left(\frac{m-1}{m}n\lambda\right)^2.$$

We can now bound:

- The maximum initial sum of variances of all players at $2\left(\frac{n\lambda(m-1)}{m}\right)^2$
- The maximum contribution to the sum of variances of all players from adding actions to the relevant set at $4n\lambda^2(\log(m-1)+1)$
- The maximum contribution to the sum of variances of all players from purifying the equilibrium at $\left(\frac{m-1}{m}n\lambda\right)^2$

Adding these together, the maximum sum of variances of actions in the relevant sets of players in $\mathbf{p}^{(n)}$ is

$$\left(\frac{m-1}{m}n\lambda\right)^2 + 4n\lambda^2(\log(m-1)+1) + 2\left(\frac{n\lambda(m-1)}{m}\right)^2$$

$$= 3\left(\frac{m-1}{m}n\lambda\right)^2 + 4n\lambda^2(\log(m-1)+1))$$
$$< 8n^2\lambda^2\log(3m)$$

for $m > 2$.

Step 3: Action profile to pure equilibrium Finally, for any value of δ_1, if any player has regret at least δ_1, the variance of their relevant actions must be at least

$$\frac{\delta_1^2}{2m} \tag{5}$$

(there must be at least a pair of relevant actions that are a distance of δ_1 apart to achieve this regret, so the lowest-variance option is realized when all remaining actions yield regret $\delta_1/2$). So allow every player with regret greater than δ_1 to instead change to playing their best response (there will be at most

$$\frac{16n^2\lambda^2 m\log(3m)}{\delta_1^2}$$

such players). As each of these players can add at most 2λ to the regret of any other player (the payoff of the best response could increase by λ while that of the chosen action could decrease by λ), the maximum regret any player can experience is

$$\delta_1 + 2\lambda\frac{16n^2\lambda^2 m\log(3m)}{\delta_1^2}.$$

Optimizing for δ_1, this value is at most

$$6\lambda\sqrt[3]{n^2 m\log(3m)}. \quad \square$$

3.2. *Hardness for* PPAD

3.2.1. *The induced population game*

This section introduces the approach that will be used to show PPAD-hardness. The technique involves artificially converting a general game into a Lipschitz game by treating every player i as a collection of many different players, each with a say in i's strategy. This reduction was used by [2] in an alternative proof of Nash's Theorem, and by [4] to upper bound the support size of ϵ-ANEs. More recently, [19] used a query-efficient version of this reduction to lower-bound the query complexity of computing ϵ-PNEs of general Lipschitz games.

Definition 12. Given a game G with payoff function $\mathbf{u}$, we define the *population game* induced by G, $G' = g_G(L)$ with payoff function $\mathbf{u}'$ in which every player i is replaced by a population of L players - v^i_ℓ for $\ell \in [L]$ - each playing G against the aggregate behavior of the other $n-1$ populations. More precisely, for $\mathbf{p}'$ a mixed profile of G',

$$u'_{v^i_\ell}(\mathbf{p}') = u_i\left(p'_{v^i_\ell}, \mathbf{p}_{-i}\right)$$

where

$$p_{i'} = \frac{1}{L}\sum_{\ell=1}^{L} p'_{v^{i'}_\ell}$$

for all $i' \neq i$.

Remark 5. Note that, regardless of the Lipschitz parameter of G, the induced population game $G' = g_G(L)$ is $\frac{1}{L}$-Lipschitz.

Population games date back even to Nash's thesis [29], in which he uses them to justify the consideration of mixed equilibria as a solution concept. To date, the reduction to the induced population game has been focused on proofs of existence. We show that the reduction can also be used to obtain the first PPAD-hardness result for a *pure* Nash equilibrium problem for a class of non-Bayesian games with discrete action spaces.

Remark 6. Note that any ϵ-PNE of $g_G(L)$ induces a $\frac{1}{L}$-uniform[2] ϵ-WSNE (and thus ϵ-ANE) of G in which each player in G plays the aggregate behavior of their population.

[2] a k-uniform mixed strategy is one in which each action of each player is assigned a probability that is a discrete multiple of k.

3.2.2. Proof of hardness

In Theorem 5, we only consider binary-action games (clearly the hardness results can be extended to games with more actions, as "dummy" actions can be added for each player that do not induce additional equilibria).

Theorem 5. *For $\epsilon < 0.088$ and $\alpha \in (\frac{1}{2}, 1)$, $(2, \epsilon, \epsilon n^{-\alpha})$-PURELPG is PPAD-hard (note that if $\alpha \leq \frac{1}{2}$ the pure equilibrium guarantee of [2] does not hold).*

Proof of Theorem 5. By Theorem 2, when $\epsilon < 0.088$ it is PPAD-hard to find ϵ-ANEs of polymatrix games (we may assume for simplicity that $\frac{1}{\epsilon} \in \mathbb{N}$). We reduce the problem of computing an ϵ-ANE of polymatrix game G to finding an ϵ-PNE of Lipschitz polymatrix game G' as follows. WLOG assume $\alpha = 1 - \frac{1}{k}$ for some integer $k > 2$, and consider the induced population game $G' = g_G(L)$ for $L = n^{k-1}/\epsilon^k$. This game has $N = (n/\epsilon)^k$ players and is $\epsilon N^{-(k-1)/k}$-Lipschitz. Thus (for large enough n) it is guaranteed to have an ϵ-PNE. Furthermore, G' is still a polymatrix game, as the payoff to any player v^i_ℓ is simply the sum of the payoffs from $N - L$ games played with the players $v^{i'}_\ell$ for $i' \neq i$.

Now, because the payoffs of any action profile of G' can be derived from payoffs of mixed strategy profiles of G (which can be computed in polynomial time), this entire reduction occurs in polynomial time. Thus, whenever $\epsilon < 0.088$ and $\lambda(n) = \epsilon n^{-(k-1)/k}$, $(2, \epsilon, \lambda)$-PURELPG is PPAD-hard. □

Combining Theorems 4 and 5 yields Theorem 3. In fact, by scaling the payoffs in Theorem 3 as described in Remark 3, we obtain:

Corollary 1. *Fix some constant $\alpha \in (\frac{2}{3}, 1)$. For any constant $m \geq 2$, non-increasing function $\epsilon(n) = \frac{1}{\text{poly}(n)}$, and $\lambda(n) = \Theta(\epsilon n^{-\alpha})$, (m, ϵ, λ)-PURELPG is PPAD-complete.*

4. Randomized PPAD

As noted above, (m, ϵ, λ)-PURELPG and the more general version relaxing the polymatrix constraint to any game with concisely-representable Lipschitz payoff functions seems to naturally belong to some randomized version of PPAD. However, unlike classes such as P or BPP, we have no real definition of a PPAD algorithm. As such, it's difficult to define randomized PPAD analogously to how the definition of BPP is often derived from the definition of P.

We certainly have other options to explore. In the coming sections we discuss a few such options and how they relate to each other. These definitions are based on the definition of search-BPP presented in [23]:

Definition 13 *(*search-BPP*).* Take a search problem defined by relation R. If there exists a polynomial-time randomized algorithm A such that

$$\Pr[R(x, A(x)) = 1] \geq \frac{2}{3}$$

and a polynomial-time randomized algorithm B such that

$$R(x, A(x)) = 0 \rightarrow \Pr[B(x, A(x)) = 0] \geq \frac{2}{3}, \qquad \Pr[B(x, A(x)) = 1] \geq \frac{1}{2}$$

and whenever there is no y such that $R(x, y) = 1$,

$$\Pr[A(x) = \perp] \geq \frac{2}{3},$$

then $R \in$ search-BPP (where the probability is taken over the randomness of A and B).

4.1. Definition of RPPAD via mapping reductions

This definition is the simplest, and is a natural extension to the standard PPAD class.

Definition 14 *(*RPPAD*).* Take a total search problem defined by relation R. If there exists a polynomial-time computable randomized function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ and deterministic algorithm A solving END OF LINE such that

$$\Pr[R(x, A(f(x))) = 1] \geq \frac{2}{3}$$

for any $x \in \{0,1\}^*$, as well as a polynomial-time randomized algorithm B such that

$$R(x, A(f(x))) = 0 \rightarrow \Pr[B(x, A(f(x))) = 0] \geq \frac{2}{3}, \qquad \Pr[B(x, A(f(x))) = 1] \geq \frac{1}{2}$$

then $R \in$ RPPAD (where the probability is taken over the randomness of f and B).

4.2. Definition of BPPPAD via Turing reductions

The next definition we present of "randomized PPAD" is, at first glance, a superset of the mapping reduction-based definition above.

Definition 15 *(BPPPAD).* We present here the definition of BPPPAD = search-BPP$^{\text{PPAD}}$. Take a total search problem defined by relation R. If there exists a polynomial-time randomized algorithm A with oracle access to END OF LINE such that

$$\Pr[R(x, A(x)) = 1] \geq \frac{2}{3}$$

for every $x \in \{0,1\}^*$ and a polynomial-time randomized algorithm B such that

$$R(x, A(x)) = 0 \rightarrow \Pr[B(x, A(x)) = 0] \geq \frac{2}{3}, \qquad \Pr[B(x, A(x)) = 1] \geq \frac{1}{2}$$

then $R \in$ BPPPAD (where the probability is taken over the randomness of A and B).

Remark 7. Note that both of the above definitions support amplification of the correctness probability, as

1. First, the correctness of decision algorithm B can be amplified.
2. With that completed, the algorithm can be run an appropriate number of times such that, with appropriately high probability, one of the runs outputs a correct answer that is accepted by B.

4.3. Pure Nash equilibria

To begin this section, we point out the following equivalence:

Theorem 6. RPPAD = BPPPAD.

This theorem is the randomized analogue of [6] (which shows that PPAD is closed under Turing reductions), and we provide a proof and example below:

Proof. Obviously, RPPAD $\subseteq$ BPPPAD, so what remains is to prove that BPPPAD $\subseteq$ RPPAD. In order to do so, we must take a relation $R \in$ BPPPAD solved by algorithm A and present an RPPAD algorithm B that mimics its behavior.

The essence of this proof consists of condensing the (polynomially-many) PPAD oracle calls into a single call that itself encodes much of the logic of algorithm A. In order to reduce to the standard (deterministic) version of this proof, we will also fix a random string r and show that

$$R(x, A(x, r)) = R(x, B(x, r))$$

for all $x \in \{0,1\}^*$.

So assume that, on input $x \in \{0,1\}^n$ and $r \in \{0,1\}^{\ell(n)}$ (for some polynomial ℓ), A makes $p(n)$ END OF LINE queries for some polynomial p (note that, because the END OF LINE queries still return deterministically, the entire sequence of queries is uniquely determined by a given value for r).[3] The i^{th} query, denoted $A_i(x, r)$, consists of a successor circuit S_i and a predecessor circuit P_i, each of size at most $t(n)$ acting on inputs (and outputting strings) of size at most $q(n)$ for polynomials t, q. To successfully convert A to B, we need to construct circuits S and P (and an interpretation function g) satisfying the following constraints:

- $P(0) = 0$
- $S(0) \neq 0$
- $y \neq P(S(y)) \rightarrow g(y) = A(x, r)$
- $(y \neq 0 \wedge y \neq S(P(y)) \rightarrow g(y) = A(x, r)$

In other words, we should be able to use g to interpret the output of this single instance as a solution to R on instance x. Note that it suffices to describe polynomial-time algorithms for S and P, since P $\subseteq$ P/poly.

So consider strings y of the form

$$a_1 b_1 \ldots a_{p(n)} b_{p(n)}$$

where

[3] We will assume there are always exactly $p(n)$ queries, as if there are fewer we can simply pad with trivial queries.

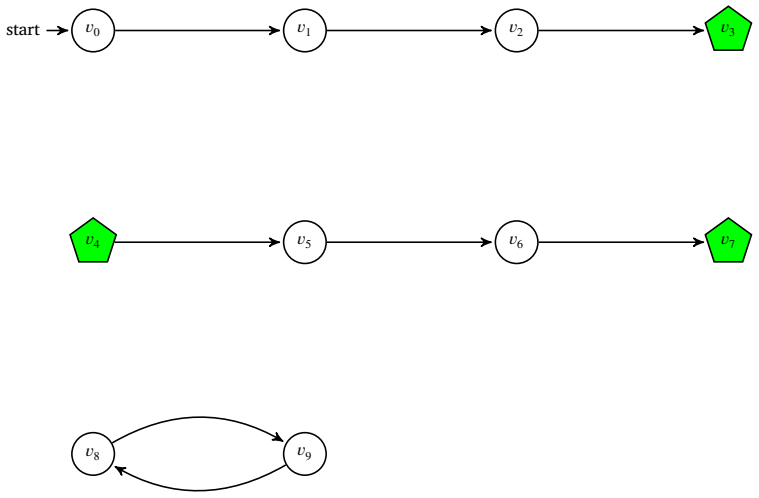

Fig. 1. END OF LINE instance to be queried three times. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

$a_1, \ldots, a_{p(n)} \in \{0,1\}^{t(n)}, b_1, \ldots, b_{p(n)} \in \{0,1\}^{q(n)}$.

Each a_i will represent an instance $A_i(x,r)$ encoding functions S_i and P_i. Each b_i will represent a solution to the instance of a_i, or (in intermediate steps) the working space in which a solution is being computed.

We consider a string y to be well-formed if and only if it satisfies both of the following properties:

1. For any nonzero a_i, $A_i(x,r) = a_i$ when simulating $A(x,r)$ by providing b_j as the answer to oracle call $A_j(x,r)$ for all $j < i$ (because A can be simulated in polynomial-time when the oracle answers are provided, this too is a polynomial-time checkable property).
2. For any nonzero a_i and any $j < i$, b_j is a solution to query a_j (because END OF LINE $\in$ **FNP** this is a polynomial-time checkable property).

For any ill-formed y (not satisfying both of the above), we will define $S(y) = P(y) = y$ to effectively remove these strings from consideration.

Now we define S and P on the remaining strings. Note that, while typically S is a "successor" circuit and P is a "predecessor", these roles do occasionally reverse. This is required to ensure that non-invertible strings of both functions are actually ultimately solutions to our entire query, and not just solutions to the first instance.

In order to define the functions formally, we present the following example. For simplicity, assume that $A(x,r)$ performs three oracle calls, all querying the same END OF LINE instance (we omit the a_i's as they should behave as described above and only include the b_i's) (Fig. 1).

Figs. 2–16 show the resulting graph from combining these three queries. Solutions are represented by green pentagons, blue triangles, teal octagons, and cyan 5-pointed stars and cases of concern represented by red 10-pointed stars (these cases are the reason that sometimes F and G must switch roles).

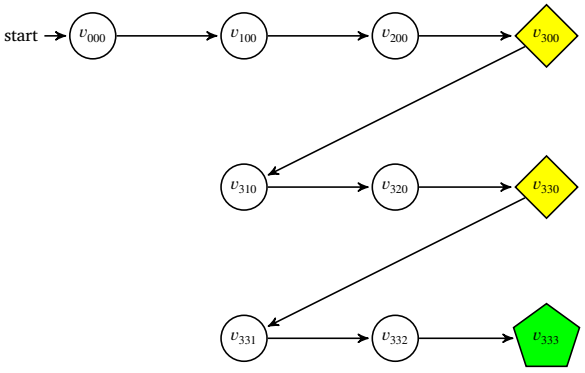

Fig. 2. The standard END OF LINE solution.

Any other string is not well-formed and is assigned a self-loop.

Essentially, Figs. 2–16 illustrate the need to ensure that the predecessor of a "4" in any query prior to the final one needs to exist and move smoothly into the next query. Consider any line in query i that does not contain "0": we will paste a forwards copy of query $i+1$ stemming from the end of the line, and a backwards copy of query $i+1$ stemming from the beginning of the line (for example, see rows 2 and 4 of Fig. 3). Thus the red 10-pointed star nodes are no longer solutions: they are instead the middle of paths for which each solution solves *all* queries. Note that there are $3^3 = 27$ possible solutions to this joint query: 3 for each index ("3", "4", and "7"). Furthermore, each well-defined triplet z is either a solution or admits a single consistent value for $S(z)$ and $P(z)$.

So, we define S and P below. Because we assume that the strings are well-formed (otherwise we have already defined their behaviors), this will be relatively simple. Take an input z to S. First, confirm this string is well-formed by performing the following verification (for simplicity we assume that 0 is not a well-formed END OF LINE instance, though this can be easily handled):

Listing 1.2. Check well-formed strings.

```
For i=1 To p(n):
    Run A(x,r) until query i;
    If A_i(x,r)==a_i:
        If (a_i,b_i)∈END OF LINE:
            Continue;
        If i==p(n) Or a_{i+1}==0:
            Return True;
        Return False;
    Else If a_i==0:
        Return True;
    Return False;
```

In other words, ensure that the two properties for well-formed strings (described above) are satisfied, i.e.:

1. The strings a_i are exactly the queries that $A(x,r)$ makes (up to this point)
2. The strings b_i are valid solutions to the instances a_i (up to this point)

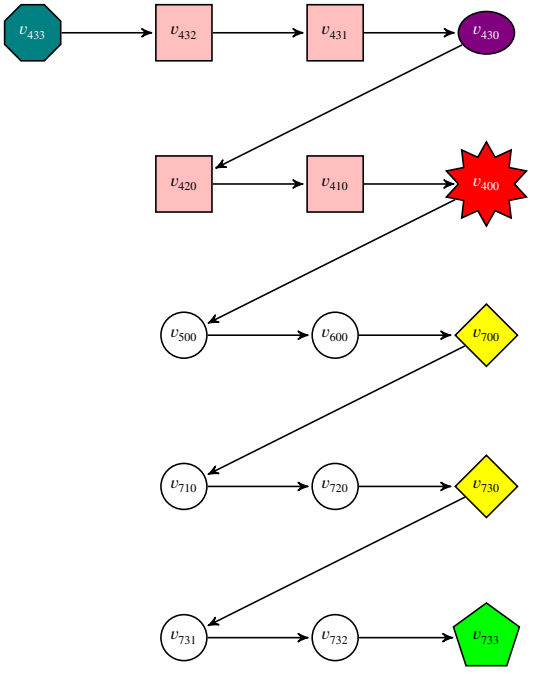

Fig. 3. The END OF LINE solution returning an end of the alternate line for the first query.

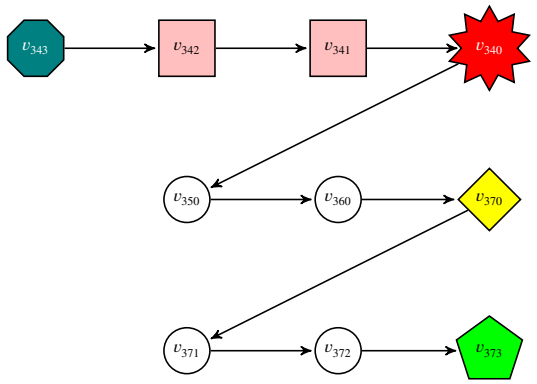

Fig. 4. The END OF LINE solution returning an end of the alternate line for the second query.

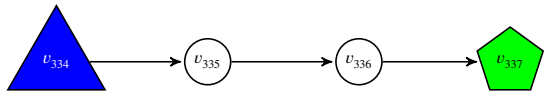

Fig. 5. The END OF LINE solution returning an end of the alternate line for the third query.

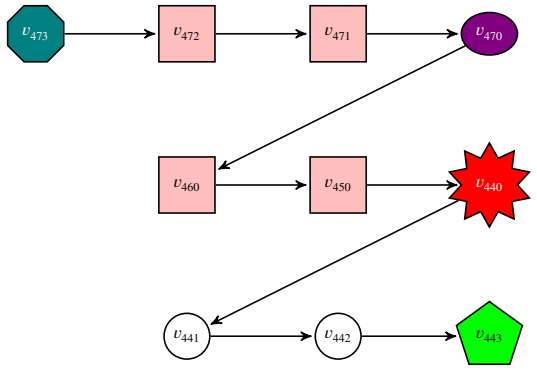

Fig. 6. The END OF LINE solution returning the *start* of the alternate line for the first query and an end of the alternate line for the second query.

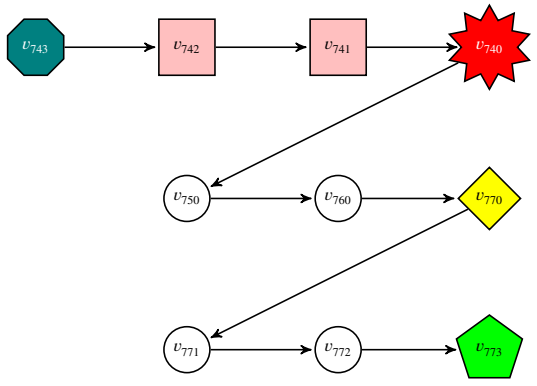

Fig. 7. The END OF LINE solution returning the *end* of the alternate line for the first query and an end of the alternate line for the second query.

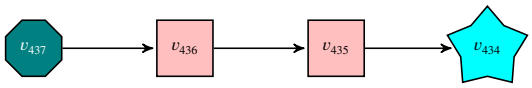

Fig. 8. The END OF LINE solution returning the *start* of the alternate line for the first query and an end of the alternate line for the third query.

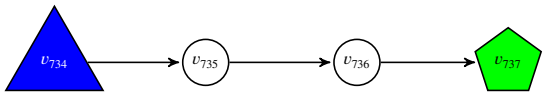

Fig. 9. The END OF LINE solution returning the *end* of the alternate line for the first query and an end of the alternate line for the third query.

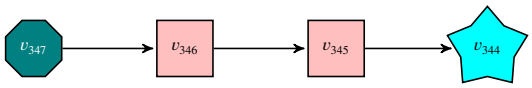

Fig. 10. The END OF LINE solution returning the *start* of the alternate line for the second query and an end of the alternate line for the third query.

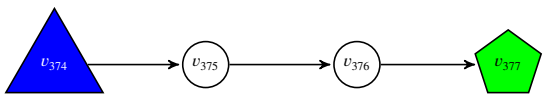

Fig. 11. The END OF LINE solution returning the *end* of the alternate line for the second query and an end of the alternate line for the third query.

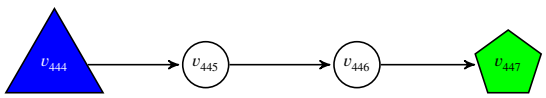

Fig. 12. The END OF LINE solution returning the *start* of the alternate line for the first and second queries and an end of the alternate line for the third query.

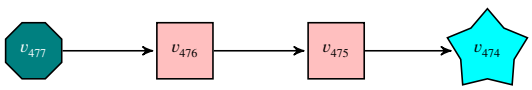

Fig. 13. The END OF LINE solution returning the *start* of the alternate line for the first query, the *end* of the alternate line for the second query, and an end of the alternate line for the third query.

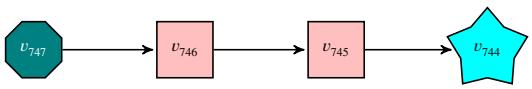

Fig. 14. The END OF LINE solution returning the *end* of the alternate line for the first query, the *start* of the alternate line for the second query, and an end of the alternate line for the third query.

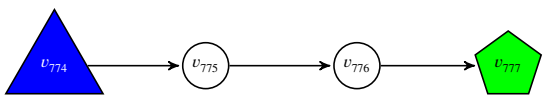

Fig. 15. The END OF LINE solution returning the *end* of the alternate line for the first and second queries and an end of the alternate line for the third query.

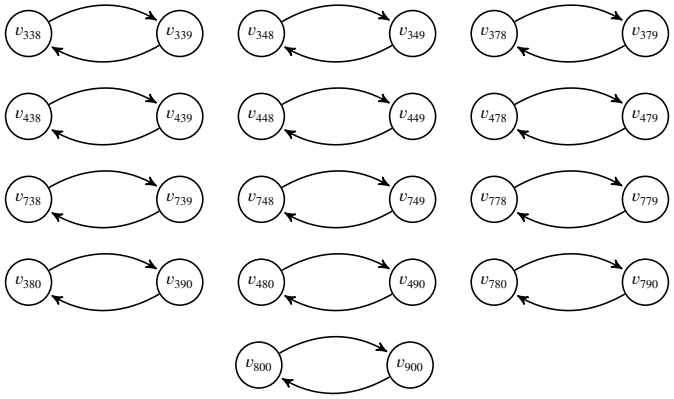

Fig. 16. The remaining well-formed strings involving nodes on the loop.

If the above is true, then take i to be the query we are currently working on (note that this can also be easily calculated from a well-formed string). The following algorithm is encoded as circuit S. The colors of specific lines correspond to the nodes at which they apply in the example above:

Listing 1.3. Definition of circuit S.

```
1  If i==1 Or an even number of {b_1,...,b_{i-1}} are beginnings of line:
2      If b_i is end of line in a_i:
3          If i == p(n):
4              Return z;
5          Return A_{i+1}(x,r);
6      Return S_i(b_i);
7  If b_i is beginning of line in a_i:
8      If i == p(n):
9          Return z;
10     Return A_{i+1}(x,r);
11 Return P_i(b_i);
```

- Line 4 describes when we have found an end of line solution to the final query.
- Line 5 describes when we have found an end of line solution to an intermediate query.
- Line 6 describes the standard case - we have not yet found a solution and simply use the successor function from instance a_i.
- Line 9 describes when we have found a *beginning* of line solution to the final query (with an odd number of *beginnings* of lines found prior).
- Line 10 describes when we have found a *beginning* of line solution to an intermediate query (it is at these nodes that the successor and predecessor circuits swap roles).
- Line 11 describes when we have not yet found a solution and (because the successor and predecessor circuits are currently swapped) we use the predecessor function from the instance a_i.

The circuit P is defined very similarly:

Listing 1.4. Definition of circuit P.

```
1  If i==1 Or an even number of {b_1,...,b_{i-1}} are beginnings of line:
2      If b_i is beginning of line in a_i:
3          If i == p(n):
4              Return z;
5          Return A_{i+1}(x,r);
6      Return P_i(b_i);
7  If b_i is end of line in a_i:
8      If i == p(n):
9          Return z;
10     Return A_{i+1}(x,r);
11 Return S_i(b_i);
```

In addition to the descriptions already associated with circuit S:

- Line 4 describes when we have found a beginning of line solution to the final query.
- Line 9 describes when we have found an *end* of line solution to the final query (with an odd number of *beginnings* of lines found prior).
- Line 10 describes when we have found an *end* of line solution to an intermediate query.

Finally, g is simply defined by the remainder of algorithm $A(x,r)$ after the final query. Algorithm B simply queries the instance of END OF LINE with circuits S and P and applies g to the output.

Let's not lose sight of the fact that we are proving the equivalence of *randomized* mapping and Turing reductions. The key insight here is that the reduction itself depends on the random string. So, for a given r, the output of $B(x,r)$ is identical to the output of $A(x,r)$. This completes the proof. □

Theorem 7. *For a fixed $m \in \mathbb{N}$, consider the problem of finding ϵ-PNEs of n-player, m-action, λ-Lipschitz games presented in the form of payoff circuits (allowed to err if the game is not λ-Lipschitz) where*

$$\epsilon = O(\lambda\sqrt{n\log n}).$$

This problem is contained in BPPPAD.

This is proven in two steps:

Lemma 7. *For a fixed $m \in \mathbb{N}$, consider the problem of finding* mixed *approximate Nash equilibria of n-player, m-action games presented in the form of payoff circuits. This problem is contained in* BPPPAD.

Remark 8. Note that this problem is not covered by [10] as (a) the input variable is the number of players rather than the number of actions and (b) the input is provided as a circuit, not a complete list of payoffs.

Proof. Recall from the proof of Lemma 3 that, in order to successfully find a mixed Nash equilibrium in PPAD, we need efficient access to mixed strategy payoffs. More specifically, we need to successfully compute an $O(\epsilon)$-approximation of these payoffs in time polynomial in n, m, and $\frac{1}{\epsilon}$ (we already assume ϵ is inverse polynomial so we need not worry about it). [20] tells us that we can successfully replace such a query with

$$\frac{64m^2}{\epsilon^2}\log\left(\frac{8n}{\eta}\right)$$

profile queries and be correct with probability at least $1-\eta$. In particular, for any polynomial $p(n,m,\frac{1}{\epsilon})$, we can guarantee an ϵ-approximation for any given payoff with probability $2^{-p(n,m,\frac{1}{\epsilon})}$ making a polynomial number of profile queries. Since we only care about exponentially many possible mixed profiles, we can ensure all queries needed by the PPAD algorithm are ϵ-approximations with failure probability inverse polynomial in n, m, and $\frac{1}{\epsilon}$ via a union bound.

We can construct the required algorithm B confirming the equilibrium in the exact same way. □

From here, we simply need to show that a random realization of this mixed equilibrium is a pure equilibrium.

Proof of Theorem 7. This is similar to the work of [2], but starts with an approximate equilibrium rather than an exact one. It is very difficult to adapt the proof in this way, but much more feasible when starting from a WSNE. As such, we maintain Step 1 of the algorithm in the proof of Lemma 6. In doing so, we can start from an ϵ_1-WSNE $\mathbf{p}^*$ for

$$\epsilon_1 = \lambda\sqrt{n}.$$

The rest of the proof closely resembles that of [2]: defining

$$E_{i,h} = [m] \times \left\{\mathbf{a}_{-i} \in [m]^{n-1} : \left|u_i(h,\mathbf{a}_{-i}) - u_i(h,\mathbf{p}^*_{-i})\right| \le \frac{\lambda}{2}\sqrt{8n\log(2n)}\right\}$$

and applying their concentration inequality,

$$\Pr_{\mathbf{a}\sim\mathbf{p}^*}[\mathbf{a} \notin E_{i,h}] \le 2\exp\left(-\frac{\left(\frac{\lambda}{2}\sqrt{8n\log(2n)}\right)^2}{2(n-1)\lambda^2}\right) < 2\exp(-\log(2n)) = \frac{1}{2n}$$

so there must exist an action profile $\mathbf{a}^*$ in the support of $\mathbf{p}^*$ such that no deviation by any player to any action affects their payoff by more than $O(\lambda\sqrt{8n\log(2n)})$ which can be found with constant probability as a random realization of $\mathbf{p}^*$. As such, for any player i and deviation d,

$$\begin{aligned} u_i(d, \mathbf{a}^*_{-i}) &\leq u_i(d, \mathbf{p}^*_{-i}) + \lambda\sqrt{8n\log(2n)} \\ &\leq u_i(a^*_i, \mathbf{p}^*_{-i}) + \lambda\sqrt{8n\log(2n)} + \lambda\sqrt{n} \\ &\leq u_i(\mathbf{a}^*) + 3\lambda\sqrt{8n\log(2n)}) \end{aligned}$$

where the first inequality comes from the fact that player i's deviation to d affects their payoff by at most $\lambda\sqrt{8n\log(2n)}$), the second from the fact that $\mathbf{p}^*$ is a $\lambda\sqrt{n}$-WSNE, and the third from the fact that player i's deviation to a^*_i once again affects their payoff by at most $\lambda\sqrt{8n\log(2n)}$). As such, with constant probability, a random realization of our ϵ_1-WSNE is an $O(\lambda\sqrt{n\log n})$-PNE.

Finally, it can be confirmed in deterministic polynomial time that this profile is an equilibrium (since we can deterministically compute the payoffs from this action profile and compare them to all nm individual deviations), so the required decision algorithm B is easy to construct. □

5. Discussion

Our PPAD-hardness result is rather strongly negative, since Lipschitz polymatrix games are quite a restricted subset of either Lipschitz or polymatrix games in general. There may be scope to broaden the (ϵ, λ) values for which Lipschitz games are hard to solve. On the other hand, there are more positive results for computing approximate Nash equilibria of Lipschitz games (see, for example, [20]), with further scope for progress.

In particular, Theorem 4 is unable to place the problem of finding ϵ-PNEs of λ-Lipschitz polymatrix games in PPAD for the total range of values guaranteed by [2] (while they guarantee existence for $\epsilon \geq \lambda\Omega(\sqrt{n\log n})$, the above successfully computes equilibria when $\epsilon \geq \lambda\Omega\left(n^{2/3}\right)$). It would be of future interest to determine if this is a characteristic of our choice of algorithm/analysis, or an indication of a barrier to the complete derandomization of computing these equilibria.

Furthermore, it may even be the case that Lipschitz *polymatrix* games, like anonymous Lipschitz games ([12]), guarantee pure equilibria for a smaller value of ϵ. It would be interesting to determine how close to a complete derandomization we are able to achieve. There are obvious obstacles to finding a deterministic lower bound asymptotically stronger than that of [2], as such a discovery would resolve the P vs. BPP question, presenting a problem solvable in randomized polynomial time but not in deterministic polynomial time (this is not only a major open question in complexity theory, but it is believed by many that P = BPP and no such problem exists).

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Matthew Katzman reports a relationship with DeepMind Technologies Ltd that includes: funding grants.

References

[1] S. Arora, B. Barak, Computational Complexity - A Modern Approach, Cambridge University Press, 2009, http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264.

[2] Y. Azrieli, E. Shmaya, Lipschitz games, Math. Oper. Res. 38 (2) (2013) 350–357, https://doi.org/10.1287/moor.1120.0557.

[3] Y. Babichenko, Best-reply dynamics in large binary-choice anonymous games, Games Econ. Behav. 81 (2013) 130–144, https://doi.org/10.1016/j.geb.2013.04.007.

[4] Y. Babichenko, Small support equilibria in large games, CoRR, arXiv:1305.2432 [abs], 2013, https://arxiv.org/abs/1305.2432.

[5] D.P. Bertsekas, J.N. Tsitsiklis, Introduction to Probability, vol. 1, Athena Scientific, 2002.

[6] S.R. Buss, A.S. Johnson, Propositional proofs and reductions between NP search problems, Ann. Pure Appl. Log. 163 (9) (2012) 1163–1182.

[7] Y. Cai, C. Daskalakis, On minmax theorems for multiplayer games, in: D. Randall (Ed.), Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, SIAM, 2011, pp. 217–234.

[8] X. Chen, X. Deng, S. Teng, Settling the complexity of computing two-player Nash equilibria, J. ACM 56 (3) (2009) 14.

[9] R. Cummings, M.J. Kearns, A. Roth, Z.S. Wu, Privacy and truthful equilibrium selection for aggregative games, in: Web and Internet Economics - 11th International Conference, WINE, in: LNCS, vol. 9470, Springer, 2015, pp. 286–299.

[10] C. Daskalakis, P.W. Goldberg, C.H. Papadimitriou, The complexity of computing a Nash equilibrium, Commun. ACM 52 (2) (2009) 89–97, https://doi.org/10.1145/1461928.1461951.

[11] C. Daskalakis, C.H. Papadimitriou, Continuous local search, in: D. Randall (Ed.), Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, SIAM, 2011, pp. 790–804.

[12] C. Daskalakis, C.H. Papadimitriou, Approximate Nash equilibria in anonymous games, J. Econ. Theory 156 (2015) 207–245.

[13] J. Deb, E. Kalai, Stability in large Bayesian games with heterogeneous players, J. Econ. Theory 157 (2015) 1041–1055, https://doi.org/10.1016/j.jet.2015.02.001.

[14] A. Deligkas, J. Fearnley, A. Hollender, T. Melissourgos, Pure-circuit: strong inapproximability for PPAD, CoRR, arXiv:2209.15149 [abs], 2022, https://doi.org/10.48550/arXiv.2209.15149.

[15] A. Deligkas, J. Fearnley, R. Savani, Tree polymatrix games are ppad-hard, in: 47th International Colloquium on Automata, Languages, and Programming, ICALP, in: LIPIcs, vol. 168, 2020, pp. 38:1–38:14.

[16] A. Deligkas, J. Fearnley, R. Savani, P.G. Spirakis, Computing approximate Nash equilibria in polymatrix games, Algorithmica 77 (2) (2017) 487–514, https://doi.org/10.1007/s00453-015-0078-7.

[17] J. Fearnley, P.W. Goldberg, A. Hollender, R. Savani, The complexity of gradient descent: CLS = PPAD ∩ PLS, in: S. Khuller, V.V. Williams (Eds.), STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021, ACM, 2021, pp. 46–59.

[18] A. Filos-Ratsikas, Y. Giannakopoulos, A. Hollender, P. Lazos, D. Poças, On the complexity of equilibrium computation in first-price auctions, in: EC '21: The 22nd ACM Conference on Economics and Computation, ACM, 2021, pp. 454–476.

[19] P.W. Goldberg, M.J. Katzman, Lower bounds for the query complexity of equilibria in Lipschitz games, in: Algorithmic Game Theory - 14th International Symposium, SAGT, in: Lecture Notes in Computer Science, vol. 12885, Springer, 2021, pp. 124–139.
[20] P.W. Goldberg, F.J. Marmolejo Cossío, Z.S. Wu, Logarithmic query complexity for approximate Nash computation in large games, Theory Comput. Syst. 63 (1) (2019) 26–53.
[21] P.W. Goldberg, A. Roth, Bounds for the query complexity of approximate equilibria, ACM Trans. Econ. Comput. 4 (4) (2016) 24.
[22] P.W. Goldberg, S. Turchetta, Query complexity of approximate equilibria in anonymous games, J. Comput. Syst. Sci. 90 (2017) 80–98.
[23] S. Goldwasser, O. Grossman, D. Holden, Pseudo-deterministic proofs, in: A.R. Karlin (Ed.), 9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, in: LIPIcs, vol. 94, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Cambridge, MA, USA, 2018, pp. 17:1–17:18.
[24] R. Gradwohl, O. Reingold, Fault tolerance in large games, Games Econ. Behav. 86 (2014) 438–457, https://doi.org/10.1016/j.geb.2013.06.007.
[25] E. Janovskaja, Equilibrium points in polymatrix games, Lith. Math. J. 8 (April 1968) 381–384, https://doi.org/10.15388/LMJ.1968.20224.
[26] D.S. Johnson, C.H. Papadimitriou, M. Yannakakis, How easy is local search?, J. Comput. Syst. Sci. 37 (1) (1988) 79–100.
[27] Z. Liu, J. Li, X. Deng, On the approximation of Nash equilibria in sparse win-lose multi-player games, in: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI, AAAI Press, 2021, pp. 5557–5565.
[28] N. Megiddo, C.H. Papadimitriou, On total functions, existence theorems and computational complexity, Theor. Comput. Sci. 81 (2) (1991) 317–324.
[29] J. Nash, Non-Cooperative Games, Ph.D. thesis, Princeton University, May 1950.
[30] J. Nash, Non-cooperative games, Ann. Math. (1951) 286–295.
[31] C.H. Papadimitriou, On the complexity of the parity argument and other inefficient proofs of existence, J. Comput. Syst. Sci. 48 (3) (1994) 498–532.
[32] C.H. Papadimitriou, B. Peng, Public goods games in directed networks, in: EC '21: The 22nd ACM Conference on Economics and Computation, ACM, 2021, pp. 745–762.
[33] A. Rubinstein, Inapproximability of Nash equilibrium, SIAM J. Comput. 47 (3) (2018) 917–959, https://doi.org/10.1137/15M1039274.
[34] G. Schoenebeck, S.P. Vadhan, The computational complexity of Nash equilibria in concisely represented games, ACM Trans. Comput. Theory 4 (2) (2012) 4, https://doi.org/10.1145/2189778.2189779.