

---

# Computing Approximate Nash Equilibria in Polymatrix Games

Argyrios Deligkas · John Fearnley ·  
Rahul Savani · Paul Spirakis

**Abstract** In an  $\epsilon$ -Nash equilibrium, a player can gain at most  $\epsilon$  by unilaterally changing his behaviour. For two-player (bimatrix) games with payoffs in  $[0, 1]$ , the best-known  $\epsilon$  achievable in polynomial time is 0.3393 [24]. In general, for  $n$ -player games an  $\epsilon$ -Nash equilibrium can be computed in polynomial time for an  $\epsilon$  that is an increasing function of  $n$  but does not depend on the number of strategies of the players. For three-player and four-player games the corresponding values of  $\epsilon$  are 0.6022 and 0.7153, respectively. Polymatrix games are a restriction of general  $n$ -player games where a player's payoff is the sum of payoffs from a number of bimatrix games. There exists a very small but constant  $\epsilon$  such that computing an  $\epsilon$ -Nash equilibrium of a polymatrix game is PPAD-hard. Our main result is that a  $(0.5 + \delta)$ -Nash equilibrium of an  $n$ -player polymatrix game can be computed in time polynomial in the input size and  $\frac{1}{\delta}$ . Inspired by the algorithm of Tsaknakis and Spirakis [24], our algorithm uses gradient descent on the maximum regret of the players. We also show that this algorithm can be applied to efficiently find a  $(0.5 + \delta)$ -Nash equilibrium in a two-player Bayesian game.

**Keywords** Approximate Nash equilibria, gradient descent, polymatrix games, Bayesian games.

---

The first author is supported by the Microsoft Research PhD sponsorship program. The second and third authors are supported by EPSRC grant EP/L011018/1, and the third author is also supported by ESRC grant ESRC/BSB/09. The work of the fourth author is supported partially by the EU ERC Project ALGAME and by the Greek THALIS action "Algorithmic Game Theory".

---

A. Deligkas · J. Fearnley · R. Savani · P. Spirakis  
Department of Computer Science, University of Liverpool, UK

P. Spirakis  
Research Academic Computer Technology Institute (CTI), Greece

## 1 Introduction

**Approximate Nash equilibria.** Nash equilibria are the central solution concept in game theory. Since it is known that computing an *exact* Nash equilibrium [9, 5] is unlikely to be achievable in polynomial time, a line of work has arisen that studies the computational aspects of approximate Nash equilibria. The most widely studied notion is of an  $\epsilon$ -*approximate Nash equilibrium* ( $\epsilon$ -Nash), which requires that all players have an expected payoff that is within  $\epsilon$  of a best response. This is an *additive* notion of approximate equilibrium; the problem of computing approximate equilibria of bimatrix games using a relative notion of approximation is known to be PPAD-hard even for constant approximations [8].

So far,  $\epsilon$ -Nash equilibria have mainly been studied in the context of two-player *bimatrix* games. A line of work [11, 10, 2] has investigated the best  $\epsilon$  that can be guaranteed in polynomial time for bimatrix games. The current best result, due to Tsaknakis and Spirakis [24], is a polynomial-time algorithm that finds a 0.3393-Nash equilibrium of a bimatrix game with all payoffs in  $[0, 1]$ .

In this paper, we study  $\epsilon$ -Nash equilibria in the context of *many-player* games, a topic that has received much less attention. A simple approximation algorithm for many-player games can be obtained by generalising the algorithm of Daskalakis, Mehta and Papadimitriou [11] from the two-player setting to the  $n$ -player setting, which provides a guarantee of  $\epsilon = 1 - \frac{1}{n}$ . This has since been improved independently by three sets of authors [3, 19, 2]. They provide a method that converts a polynomial-time algorithm that for finding  $\epsilon$ -Nash equilibria in  $(n - 1)$ -player games into an algorithm that finds a  $\frac{1}{2-\epsilon}$ -Nash equilibrium in  $n$ -player games. Using the polynomial-time 0.3393 algorithm of Tsaknakis and Spirakis [24] for 2-player games as the base case for this recursion, this allows us to provide polynomial-time algorithms with approximation guarantees of 0.6022 in 3-player games, and 0.7153 in 4-player games. These guarantees tend to 1 as  $n$  increases, and so far, no constant  $\epsilon < 1$  is known such that, for all  $n$ , an  $\epsilon$ -Nash equilibrium of an  $n$ -player game can be computed in polynomial time.

For  $n$ -player games, we have lower bounds for  $\epsilon$ -Nash equilibria. More precisely, Rubinstein has shown that when  $n$  is not a constant there exists a constant but very small  $\epsilon$  such that it is PPAD-hard to compute an  $\epsilon$ -Nash equilibrium [23]. This is quite different from the bimatrix game setting, where the existence of a quasi-polynomial time approximation scheme rules out such a lower bound, unless all of PPAD can be solved in quasi-polynomial time [22].

**Polymatrix games.** In this paper, we focus on a particular class of many-player games called *polymatrix games*. In a polymatrix game, the interaction between the players is specified by an  $n$  vertex graph, where each vertex represents one of the players. Each edge of the graph specifies a bimatrix game that will be played by the two respective players, and thus a player with degree  $d$  will play  $d$  bimatrix games simultaneously. More precisely, each player picks a

strategy, and then plays this strategy in *all* of the bimatrix games that he is involved in. His payoff is then the sum of the payoffs that he obtains in each of the games.

Polymatrix games are a class of *succinctly represented*  $n$ -player games: a polymatrix game is specified by at most  $n^2$  bimatrix games, each of which can be written down in quadratic space with respect to the number of strategies. This is unlike general  $n$ -player strategic form games, which require a representation that is exponential in the number of players.

There has been relatively little work on approximation algorithms for polymatrix games. The approximation algorithms for general games can be applied in this setting in an obvious way, but to the best of our knowledge there have been no upper bounds that are specific to polymatrix games. On the other hand, the lower bound of Rubinstein mentioned above is actually proved by constructing polymatrix games. Thus, there is a constant but very small  $\epsilon$  such that it is PPAD-hard to compute an  $\epsilon$ -Nash equilibrium [23], and this again indicates that approximating polymatrix games is quite different to approximating bimatrix games.

**Our contribution.** Our main result is an algorithm that, for every  $\delta$  in the range  $0 < \delta \leq 0.5$ , finds a  $(0.5 + \delta)$ -Nash equilibrium of a polymatrix game in time polynomial in the input size and  $\frac{1}{\delta}$ . Note that our approximation guarantee *does not depend on the number of players*, which is a property that was not previously known to be achievable for polymatrix games, and still cannot be achieved for general strategic form games.

We prove this result by adapting the algorithm of Tsaknakis and Spirakis [24] (henceforth referred to as the TS algorithm). They give a gradient descent algorithm for finding a 0.3393-Nash equilibrium in a bimatrix game. We generalise their gradient descent techniques to the polymatrix setting, and show that it always arrives at a  $(0.5 + \delta)$ -Nash equilibrium after a polynomial number of iterations.

In order to generalise the TS algorithm, we had to overcome several issues. Firstly, the TS algorithm makes the regrets of the two players equal in every iteration, but there is no obvious way to achieve this in the polymatrix setting. Instead, we show how gradient descent can be applied to a strategy profile where the regrets are not necessarily equal. Secondly, the output of the TS algorithm is either a point found by gradient descent, or a point obtained by modifying the result of gradient descent. In the polymatrix game setting, it is not immediately obvious how such a modification can be derived with a non-constant number of players (without an exponential blowup). Thus we apply a different analysis, which proves that the point resulting from gradient descent always has our approximation guarantee. It is an interesting open question whether a better approximation guarantee can be achieved when there is a constant number of players.

An interesting feature of our algorithm is that it can be applied even when players have differing degrees. Originally, polymatrix games were defined only for complete graphs [20]. Since previous work has only considered lower bounds

for polymatrix games, it has been sufficient to restrict attention to regular graphs, as in work Rubinstein [23]. However, since this paper is proving an upper bound, we must be more careful. As it turns out, our algorithm will efficiently find a  $(0.5 + \delta)$ -Nash equilibrium for all  $\delta > 0$ , no matter what graph structure the polymatrix game has.

Finally, we show that our algorithm can be applied to two-player Bayesian games. In a two-player Bayesian game, each player is assigned a type according to a publicly known probability distribution. Each player knows their own type, but does not know the type of their opponent. We show that finding an  $\epsilon$ -Nash equilibrium in these games can be reduced to the problem of finding an  $\epsilon$ -Nash equilibrium in a polymatrix game, and therefore, our algorithm can be used to efficiently find a  $(0.5 + \delta)$ -Nash equilibrium of a two-player Bayesian game.

**Related work.** An FPTAS for the problem of computing an  $\epsilon$ -Nash equilibrium of a bimatrix game does not exist unless every problem in **PPAD** can be solved in polynomial time [5]. Arguably, the biggest open question in equilibrium computation is whether there exists a PTAS for this problem. As we have mentioned, for any constant  $\epsilon > 0$ , there does exist a *quasi-polynomial*-time algorithm for computing an  $\epsilon$ -Nash equilibria of a bimatrix game, or any game with a constant number of players [22, 1], with running time  $k^{O(\log k)}$  for a  $k \times k$  bimatrix game. Consequently, in contrast to the many-player case, it is not believed that there exists a constant  $\epsilon$  such that the problem of computing an  $\epsilon$ -Nash equilibrium of a bimatrix game (or any game with a constant number of players) is **PPAD**-hard, since it seems unlikely that all problems in **PPAD** have quasi-polynomial-time algorithms. On the other hand, for multi-player games, as mentioned above, there is a small constant  $\epsilon$  such that it is **PPAD**-hard to compute an  $\epsilon$ -Nash equilibrium of an  $n$ -player game when  $n$  is not constant. One positive result we do have for multi-player games is that there is a PTAS for *anonymous games* (where the identity of players does not matter) when the number of strategies is constant [12].

Polymatrix games have played a central role in the reductions that have been used to show **PPAD**-hardness of games and other equilibrium problems [9, 5, 14, 16, 6]. Computing an *exact* Nash equilibrium in a polymatrix game is **PPAD**-hard even when all the bimatrix games played are either zero-sum games or coordination games [4]. Polymatrix games have been used in other contexts too. For example, Govindan and Wilson proposed a (non-polynomial-time) algorithm for computing Nash equilibria of an  $n$ -player game, by approximating the game with a sequence of polymatrix games [17]. Later, they presented a (non-polynomial) reduction that reduces  $n$ -player games to polymatrix games while preserving approximate Nash equilibria [18]. Their reduction introduces a central coordinator player, who interacts bilaterally with every player.

## 2 Preliminaries

We start by fixing some notation. We use  $[k]$  to denote the set of integers  $\{1, 2, \dots, k\}$ , and when a universe  $[k]$  is clear, we will use  $\bar{S} = \{i \in [k], i \notin S\}$  to denote the complement of  $S \subseteq [k]$ . For a  $k$ -dimensional vector  $x$ , we use  $x_{-S}$  to denote the elements of  $x$  with indices  $\bar{S}$ , and in the case where  $S = \{i\}$  has only one element, we simply write  $x_{-i}$  for  $x_{-S}$ .

**Polymatrix games.** An  $n$ -player polymatrix game is defined by an undirected graph  $(V, E)$  with  $n$  vertices, where every vertex corresponds to a player. The edges of the graph specify which players interact with each other. For each  $i \in [n]$ , we use  $N(i) = \{j : (i, j) \in E\}$  to denote the neighbours of player  $i$ .

Each edge  $(i, j) \in E$  specifies that a bimatrix game will be played between players  $i$  and  $j$ . Each player  $i \in [n]$  has a fixed number of pure strategies  $m_i$ , and the bimatrix game on edge  $(i, j) \in E$  will therefore be specified by an  $m_i \times m_j$  matrix  $A_{ij}$ , which gives the payoffs for player  $i$ , and an  $m_j \times m_i$  matrix  $A_{ji}$ , which gives the payoffs for player  $j$ . We allow the individual payoffs in each matrix to be an arbitrary (even negative) rational number. As we describe in the next subsection, we will rescale these payoffs so that the overall payoff to each player lies in the range  $[0, 1]$ .

### 2.1 Payoff Normalization

Before we continue, we must first discuss how the payoffs in the game are rescaled. It is common, when proving results about additive notions of approximate equilibria, to rescale the payoffs of the game. This is necessary in order for different results to be comparable. For example, all results about additive approximate equilibria in bimatrix games assume that the payoff matrices have entries in the range  $[0, 1]$ , and therefore an  $\epsilon$ -Nash equilibrium always has a consistent meaning. For the same reason, we must rescale the payoffs in a polymatrix in order to give a consistent meaning to an  $\epsilon$ -approximation.

An initial, naive, approach would be to specify that each of the individual bimatrix games has entries in the range  $[0, 1]$ . This would be sufficient if we were only interested in polymatrix games played on either complete graphs or regular graphs. However, in this model, if the players have differing degrees, then they also have differing maximum payoffs. This means that an additive approximate equilibrium must pay more attention to high degree players, as they can have larger regrets.

One solution to this problem, which was adopted in the conference version of this paper [13], is to rescale according to the degree. That is, given a polymatrix game where each bimatrix game has payoffs in the range  $[0, 1]$ , if a player has degree  $d$ , then each of his payoff matrices is divided by  $d$ . This transformation ensures that every player has regret in the range  $[0, 1]$ , and therefore low degree players are not unfairly treated by additive approximations.

However, rescaling according to the degree assumes that each bimatrix game actually uses the full range of payoffs between  $[0, 1]$ . In particular, some bimatrix games may have minimum payoff strictly greater than 0, or maximum payoff strictly less than 1. This issue arises, in particular, in our application of two-player Bayesian games. Note that, unlike the case of a single bimatrix game, we cannot fix this by rescaling individual bimatrix games in a polymatrix game, because we must maintain the relationship between the payoffs in all of the bimatrix games that a player is involved in.

To address this, we will rescale the games so that, for each player, the minimum possible payoff is 0, and the maximum possible payoff is 1. For each player  $i$ , we denote by  $U$  the maximum payoff he can obtain, and by  $L$  the minimum payoff he can obtain. Formally:

$$U_i := \max_{p \in [m_i]} \left( \sum_{j \in N(i)} \max_{q \in [m_j]} (A_{ij}(p, q)) \right),$$

$$L_i := \min_{p \in [m_i]} \left( \sum_{j \in N(i)} \min_{q \in [m_j]} (A_{ij}(p, q)) \right).$$

Then, for all  $i$  and all  $j \in N(i)$  we will apply the following transformation, which we call  $T(\cdot)$ , to all the entries  $z$  of payoff matrices  $A_{ij}$ :

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

Observe that, since player  $i$ 's payoff is the sum of  $d(i)$  many bimatrix games, it must be the case that after transforming the payoff matrices in this way, player  $i$ 's maximum possible payoff is 1, and player  $i$ 's minimum possible payoff is 0. For the rest of this paper, we will assume that the payoff matrices given by  $A_{ij}$  are rescaled in this way.

## 2.2 Approximate Nash Equilibria

**Strategies.** A *mixed strategy* for player  $i$  is a probability distribution over player  $i$ 's pure strategies. Formally, for each positive integer  $k$ , we denote the  $(k - 1)$ -dimensional simplex by  $\Delta_k := \{x : x \in \mathbb{R}^k, x \geq 0, \sum_{i=1}^k x_i = 1\}$ , and therefore the set of strategies for player  $i$  is  $\Delta_{m_i}$ . For each mixed strategy  $x \in \Delta_m$ , the *support* of  $x$  is defined as  $\text{supp}(x) := \{i \in [m] : x_i \neq 0\}$ , which is the set of strategies played with positive probability by  $x$ .

A *strategy profile* specifies a mixed strategy for every player. We denote the set of mixed strategy profiles as  $\Delta := \Delta_{m_1} \times \dots \times \Delta_{m_n}$ . Given a strategy profile  $\mathbf{x} = (x_1, \dots, x_n) \in \Delta$ , the payoff of player  $i$  under  $\mathbf{x}$  is the sum of the payoffs that he obtains in each of the bimatrix games that he plays. Formally, we define:

$$u_i(\mathbf{x}) := x_i^T \sum_{j \in N(i)} A_{ij} x_j. \quad (1)$$

We denote by  $u_i(x'_i, \mathbf{x})$  the payoff for player  $i$  when he plays  $x'_i$  and the other players play according to the strategy profile  $\mathbf{x}$ . In some cases the first argument will be  $x_i - x'_i$  which may not correspond to a valid strategy for player  $i$  but we still apply the equation as follows:

$$u_i(x_i - x'_i, \mathbf{x}) := x_i^T \sum_{j \in N(i)} A_{ij} x_j - x_i'^T \sum_{j \in N(i)} A_{ij} x_j = u_i(x_i, \mathbf{x}) - u_i(x'_i, \mathbf{x}).$$

**Best responses.** Let  $v_i(\mathbf{x})$  be the vector of payoffs for each pure strategy of player  $i$  when the rest of players play strategy profile  $\mathbf{x}$ . Formally,

$$v_i(\mathbf{x}) = \sum_{j \in N(i)} A_{ij} x_j.$$

For each vector  $x \in R^m$ , we define  $\text{suppmax}(x)$  to be the set of indices that achieve the maximum of  $x$ , that is, we define  $\text{suppmax}(x) = \{i \in [m] : x_i \geq x_j, \forall j \in [m]\}$ . Then the *pure best responses* of player  $i$  against a strategy profile  $\mathbf{x}$  (where only  $\mathbf{x}_{-i}$  is relevant) is given by:

$$\text{Br}_i(\mathbf{x}) = \text{suppmax} \left( \sum_{j \in N(i)} A_{ij} x_j \right) = \text{suppmax}(v_i(\mathbf{x})). \quad (2)$$

The corresponding *best response payoff* is given by:

$$u_i^*(\mathbf{x}) = \max_k \left\{ \left( \sum_{j \in N(i)} A_{ij} x_j \right)_k \right\} = \max_k \{ (v_i(\mathbf{x}))_k \}. \quad (3)$$

**Equilibria.** In order to define the exact and approximate equilibria of a polymatrix game, we first define the *regret* that is suffered by each player under a given strategy profile. The regret function  $f_i : \Delta \rightarrow [0, 1]$  is defined, for each player  $i$ , as follows:

$$f_i(\mathbf{x}) := u_i^*(\mathbf{x}) - u_i(\mathbf{x}). \quad (4)$$

The maximum regret under a strategy profile  $\mathbf{x}$  is given by the function  $f(\mathbf{x})$  where:

$$f(\mathbf{x}) := \max\{f_1(\mathbf{x}), \dots, f_n(\mathbf{x})\}. \quad (5)$$

We say that  $\mathbf{x}$  is an  $\epsilon$ -approximate Nash equilibrium ( $\epsilon$ -NE) if we have:

$$f(\mathbf{x}) \leq \epsilon,$$

and  $\mathbf{x}$  is an *exact* Nash equilibrium if we have  $f(\mathbf{x}) = 0$ .

### 3 The gradient

Our goal is to apply gradient descent to the regret function  $f$ . In this section, we formally define the gradient of  $f$  in Definition 1, and give a reformulation of that definition in Lemma 3. In order to show that our gradient descent method terminates after a polynomial number of iterations, we actually need to use a slightly modified version of this reformulation, which we describe at the end of this section in Definition 5.

Given a point  $\mathbf{x} \in \Delta$ , a *feasible direction* from  $\mathbf{x}$  is defined by any other point  $\mathbf{x}' \in \Delta$ . This defines a line between  $\mathbf{x}$  and  $\mathbf{x}'$ , and formally speaking, the direction of this line is  $\mathbf{x}' - \mathbf{x}$ . In order to define the gradient of this direction, we consider the function  $f((1 - \epsilon) \cdot \mathbf{x} + \epsilon \cdot \mathbf{x}') - f(\mathbf{x})$  where  $\epsilon$  lies in the range  $0 \leq \epsilon \leq 1$ . The gradient of this direction is given in the following definition.

**Definition 1** Given profiles  $\mathbf{x}, \mathbf{x}' \in \Delta$  and  $\epsilon \in [0, 1]$ , we define:

$$Df(\mathbf{x}, \mathbf{x}', \epsilon) := f((1 - \epsilon) \cdot \mathbf{x} + \epsilon \cdot \mathbf{x}') - f(\mathbf{x}).$$

Then, we define the gradient of  $f$  at  $\mathbf{x}$  in the direction  $\mathbf{x}' - \mathbf{x}$  as:

$$Df(\mathbf{x}, \mathbf{x}') = \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} Df(\mathbf{x}, \mathbf{x}', \epsilon).$$

This is the natural definition of the gradient, but it cannot be used directly in a gradient descent algorithm. We now show how this definition can be reformulated. Firstly, for each  $\mathbf{x}, \mathbf{x}' \in \Delta$ , and for each player  $i \in [n]$ , we define:

$$Df_i(\mathbf{x}, \mathbf{x}') := \max_{k \in \text{Br}_i(\mathbf{x})} \{ (v_i(\mathbf{x}'))_k \} - u_i(x_i, \mathbf{x}') + u_i(x_i - x'_i, \mathbf{x}). \quad (6)$$

Next we define  $\mathcal{K}(\mathbf{x})$  to be the set of players that have maximum regret under the strategy profile  $\mathbf{x}$ .

**Definition 2** Given a strategy profile  $\mathbf{x}$ , define  $\mathcal{K}(\mathbf{x})$  as follows:

$$\mathcal{K}(\mathbf{x}) := \{ i \in [n], f_i(\mathbf{x}) = f(\mathbf{x}) \} = \left\{ i \in [n], f_i(\mathbf{x}) = \max_{j \in [n]} f_j(\mathbf{x}) \right\}. \quad (7)$$

The following lemma, which is proved in Appendix A, provides our reformulation.

**Lemma 3** *The gradient of  $f$  at point  $\mathbf{x}$  along direction  $\mathbf{x}' - \mathbf{x}$  is:*

$$Df(\mathbf{x}, \mathbf{x}') = \max_{i \in \mathcal{K}(\mathbf{x})} Df_i(\mathbf{x}, \mathbf{x}') - f(\mathbf{x}).$$

In order to show that our gradient descent algorithm terminates after a polynomial number of steps, we have to use a slight modification of the formula given in Lemma 3. More precisely, in the definition of  $Df_i(\mathbf{x}, \mathbf{x}')$ , we need to take the maximum over the  $\delta$ -best responses, rather than the best responses.

We begin by providing the definition of the  $\delta$ -best responses.



**Definition 4 ( $\delta$ -best response)** Let  $\mathbf{x} \in \Delta$ , and let  $\delta \in (0, 0.5]$ . The  $\delta$ -best response set  $\text{Br}_i^\delta(\mathbf{x})$  for player  $i \in [n]$  is defined as:

$$\text{Br}_i^\delta(\mathbf{x}) := \left\{ j \in [m_i] : (v_i(\mathbf{x}))_j \geq u_i^*(\mathbf{x}) - \delta \right\}.$$

We now define the function  $Df_i^\delta(\mathbf{x}, \mathbf{x}')$ .

**Definition 5** Let  $\mathbf{x}, \mathbf{x}' \in \Delta$ , let  $\epsilon \in [0, 1]$ , and let  $\delta \in (0, 0.5]$ . We define  $Df_i^\delta(\mathbf{x}, \mathbf{x}')$  as:

$$Df_i^\delta(\mathbf{x}, \mathbf{x}') := \max_{k \in \text{Br}_i^\delta(\mathbf{x})} \left\{ (v_i(\mathbf{x}'))_k \right\} - u_i(x_i, \mathbf{x}') - u_i(x'_i, \mathbf{x}) + u_i(x_i, \mathbf{x}). \quad (8)$$

Furthermore, we define  $Df^\delta(\mathbf{x}, \mathbf{x}')$  as:

$$Df^\delta(\mathbf{x}, \mathbf{x}') = \max_{i \in \mathcal{K}(\mathbf{x})} Df_i^\delta(\mathbf{x}, \mathbf{x}') - f(\mathbf{x}). \quad (9)$$

Our algorithm works by performing gradient descent using the function  $Df^\delta$  as the gradient. Obviously, this is a different function to  $Df$ , and so we are not actually performing gradient descent on the gradient of  $f$ . It is important to note that all of our proofs are in terms of  $Df^\delta$ , and so this does not affect the correctness of our algorithm. We proved Lemma 3 in order to explain where our definition of the gradient comes from, but the correctness of our algorithm does not depend on the correctness of Lemma 3.

## 4 The algorithm

In this section, we describe our algorithm for finding a  $(0.5 + \delta)$ -Nash equilibrium in a polymatrix game by gradient descent. In each iteration of the algorithm, we must find the *direction* of steepest descent with respect to  $Df^\delta$ . We show that this task can be achieved by solving a linear program, and we then use this LP to formally specify our algorithm.

**The direction of steepest descent.** We show that the direction of steepest descent can be found by solving a linear program. Our goal is, for a given strategy profile  $\mathbf{x}$ , to find another strategy profile  $\mathbf{x}'$  so as to minimize the gradient  $Df^\delta(\mathbf{x}, \mathbf{x}')$ . Recall that  $Df^\delta$  is defined in Equation (9) to be:

$$Df^\delta(\mathbf{x}, \mathbf{x}') = \max_{i \in \mathcal{K}(\mathbf{x})} Df_i^\delta(\mathbf{x}, \mathbf{x}') - f(\mathbf{x}).$$

Note that the term  $f(\mathbf{x})$  is a constant in this expression, because it is the same for all directions  $\mathbf{x}'$ . Thus, it is sufficient to formulate a linear program in order to find the  $\mathbf{x}'$  that minimizes  $\max_{i \in \mathcal{K}(\mathbf{x})} Df_i^\delta(\mathbf{x}, \mathbf{x}')$ . Using the definition of  $Df_i^\delta$  in Equation (8), we can do this as follows.

**Definition 6 (Steepest descent linear program)** Given a strategy profile  $\mathbf{x}$ , the *steepest descent linear program* is defined as follows. Find  $\mathbf{x}' \in \Delta$ ,  $l_1, l_2, \dots, l_{|\mathcal{K}(\mathbf{x})|}$ , and  $w$  such that:

$$\begin{aligned} & \text{minimize} && w \\ & \text{subject to} && (v_i(\mathbf{x}'))_k \leq l_i && \forall k \in \text{Br}_i^\delta(\mathbf{x}), \quad \forall i \in \mathcal{K}(\mathbf{x}) \\ & && l_i - u_i(x_i, \mathbf{x}') - u_i(x'_i, \mathbf{x}) + u_i(\mathbf{x}) \leq w && \forall i \in \mathcal{K}(\mathbf{x}) \\ & && \mathbf{x}' \in \Delta. \end{aligned}$$

The  $l_i$  variables deal with the maximum in the term  $\max_{k \in \text{Br}_i^\delta(\mathbf{x})} \{(v_i(\mathbf{x}'))_k\}$ , while the variable  $w$  is used to deal with the maximum over the functions  $Df_i^\delta$ . Since the constraints of the linear program correspond precisely to the definition of  $Df_i^\delta$ , it is clear that, when we minimize  $w$ , the resulting  $\mathbf{x}'$  specifies the direction of steepest descent. For each profile  $\mathbf{x}$ , we define  $Q(\mathbf{x})$  to be the direction  $\mathbf{x}'$  found by the steepest descent LP for  $\mathbf{x}$ .

Once we have found the direction of steepest descent, we then need to move in that direction. More precisely, we fix a parameter  $\epsilon = \frac{\delta}{\delta+2}$  which is used to determine how far we move in the steepest descent direction. We will show in Section 6 that this value of  $\epsilon$  leads to a polynomial bound on the running time of our algorithm.

**The algorithm.** We can now formally describe our algorithm. The algorithm takes a parameter  $\delta \in (0, 0.5]$ , which will be used as a tradeoff between running time and the quality of approximation.

#### Algorithm 1

1. Choose an arbitrary strategy profile  $\mathbf{x} \in \Delta$ .
2. Solve the steepest descent linear program with input  $\mathbf{x}$  to obtain  $\mathbf{x}' = Q(\mathbf{x})$ .
3. Set  $\mathbf{x} := \mathbf{x} + \epsilon(\mathbf{x}' - \mathbf{x})$ , where  $\epsilon = \frac{\delta}{\delta+2}$ .
4. If  $f(\mathbf{x}) \leq 0.5 + \delta$  then stop, otherwise go to step 2.

A single iteration of this algorithm corresponds to executing steps 2, 3, and 4. Since this only involves solving a single linear programs, it is clear that each iteration can be completed in polynomial time.

The rest of this paper is dedicated to showing the following theorem, which is our main result.

**Theorem 7** *Algorithm 1 finds a  $(0.5 + \delta)$ -NE after at most  $O(\frac{1}{\delta^2})$  iterations.*

To prove Theorem 7, we will show two properties. Firstly, in Section 5, we show that our gradient descent algorithm never gets stuck in a stationary point

before it finds a  $(0.5 + \delta)$ -NE. To do so, we define the notion of a  $\delta$ -stationary point, and we show that every  $\delta$ -stationary point is at least a  $(0.5 + \delta)$ -NE, which then directly implies that the gradient descent algorithm will not get stuck before it finds a  $(0.5 + \delta)$ -NE.

Secondly, in Section 6, we prove the upper bound on the number of iterations. To do this we show that, if an iteration of the algorithm starts at a point that is not a  $\delta$ -stationary point, then that iteration will make a large enough amount of progress. This then allows us to show that the algorithm will find a  $(0.5 + \delta)$ -NE after  $O(\frac{1}{\delta^2})$  many iterations, and therefore the overall running time of the algorithm is polynomial.

## 5 Stationary points

Recall that Definition 6 gives a linear program for finding the direction  $\mathbf{x}'$  that minimises  $Df^\delta(\mathbf{x}, \mathbf{x}')$ . Our steepest descent procedure is able to make progress whenever this gradient is negative, and so a stationary point is any point  $\mathbf{x}$  for which  $Df^\delta(\mathbf{x}, \mathbf{x}') \geq 0$ . In fact, our analysis requires us to consider  $\delta$ -stationary points, which we now define.

**Definition 8 ( $\delta$ -stationary point)** Let  $\mathbf{x}^*$  be a mixed strategy profile, and let  $\delta > 0$ . We have that  $\mathbf{x}^*$  is a  $\delta$ -stationary point if for all  $\mathbf{x}' \in \Delta$ :

$$Df^\delta(\mathbf{x}^*, \mathbf{x}') \geq -\delta.$$

We now show that every  $\delta$ -stationary point of  $f(\mathbf{x})$  is a  $(0.5 + \delta)$ -NE. Recall from Definition 5 that:

$$Df^\delta(\mathbf{x}, \mathbf{x}') = \max_{i \in \mathcal{K}(\mathbf{x})} Df_i^\delta(\mathbf{x}, \mathbf{x}') - f(\mathbf{x}).$$

Therefore, if  $\mathbf{x}^*$  is a  $\delta$ -stationary point, we must have, for every direction  $\mathbf{x}'$ :

$$f(\mathbf{x}^*) \leq \max_{i \in \mathcal{K}(\mathbf{x})} Df_i^\delta(\mathbf{x}^*, \mathbf{x}') + \delta. \quad (10)$$

Since  $f(\mathbf{x}^*)$  is the maximum regret under the strategy profile  $\mathbf{x}^*$ , in order to show that  $\mathbf{x}^*$  is a  $(0.5 + \delta)$ -NE, we only have to find some direction  $\mathbf{x}'$  such that  $\max_{i \in \mathcal{K}(\mathbf{x})} Df_i^\delta(\mathbf{x}, \mathbf{x}') \leq 0.5$ . We do this in the following lemma.

**Lemma 9** *In every stationary point  $\mathbf{x}^*$ , there exists a direction  $\mathbf{x}'$  such that:*

$$\max_{i \in \mathcal{K}(\mathbf{x})} Df_i^\delta(\mathbf{x}^*, \mathbf{x}') \leq 0.5.$$

*Proof* First, define  $\bar{\mathbf{x}}$  to be a strategy profile in which each player  $i \in [n]$  plays a best response against  $\mathbf{x}^*$ . We will set  $\mathbf{x}' = \frac{\bar{\mathbf{x}} + \mathbf{x}^*}{2}$ . Then for each  $i \in \mathcal{K}(\mathbf{x})$ , we

have that  $Df_i^\delta(\mathbf{x}^*, \mathbf{x}')$ , is less than or equal to:

$$\begin{aligned}
& \max_{k \in \text{Br}_i^\delta(\mathbf{x}^*)} \left\{ \left( v_i \left( \frac{\bar{\mathbf{x}} + \mathbf{x}^*}{2} \right) \right)_k \right\} - u_i \left( x_i^*, \frac{\bar{\mathbf{x}} + \mathbf{x}^*}{2} \right) - u_i \left( \frac{\bar{x}_i + x_i^*}{2}, \mathbf{x}^* \right) + u_i(x_i^*, \mathbf{x}^*) \\
&= \frac{1}{2} \cdot \max_{k \in \text{Br}_i^\delta(\mathbf{x}^*)} \left\{ \left( v_i(\bar{\mathbf{x}} + \mathbf{x}^*) \right)_k \right\} - \frac{1}{2} \cdot u_i(x_i^*, \bar{\mathbf{x}}) - \frac{1}{2} \cdot u_i(\bar{x}_i, \mathbf{x}^*) \\
&\leq \frac{1}{2} \cdot \left( \max_{k \in \text{Br}_i^\delta(\mathbf{x}^*)} \left\{ \left( v_i(\bar{\mathbf{x}}) \right)_k \right\} + \max_{k \in \text{Br}_i^\delta(\mathbf{x}^*)} \left\{ \left( v_i(\mathbf{x}^*) \right)_k \right\} - u_i(x_i^*, \bar{\mathbf{x}}) - u_i(\bar{x}_i, \mathbf{x}^*) \right) \\
&= \frac{1}{2} \cdot \left( \max_{k \in \text{Br}_i^\delta(\mathbf{x}^*)} \left\{ \left( v_i(\bar{\mathbf{x}}) \right)_k \right\} - u_i(x_i^*, \bar{\mathbf{x}}) \right) \quad \text{because } \bar{x}_i \text{ is a b.r. to } x^* \\
&\leq \frac{1}{2} \cdot \max_{k \in \text{Br}_i^\delta(\mathbf{x}^*)} \left\{ \left( v_i(\bar{\mathbf{x}}) \right)_k \right\} \\
&\leq \frac{1}{2}.
\end{aligned}$$

Thus, the point  $\mathbf{x}'$  satisfies  $\max_{i \in \mathcal{K}(\mathbf{x})} Df_i^\delta(\mathbf{x}^*, \mathbf{x}') \leq 0.5$ .  $\square$

We can sum up the results of the section in the following lemma.

**Lemma 10** *Every  $\delta$ -stationary point  $\mathbf{x}^*$  is a  $(0.5 + \delta)$ -Nash equilibrium.*

## 6 The time complexity of the algorithm

In this section, we show that Algorithm 1 terminates after a polynomial number of iterations. Let  $\mathbf{x}$  be a strategy profile that is considered by Algorithm 1, and let  $\mathbf{x}' = Q(\mathbf{x})$  be the solution of the steepest descent LP for  $\mathbf{x}$ . These two profiles will be fixed throughout this section.

We begin by proving a technical lemma that will be crucial for showing our bound on the number of iterations. To simplify our notation, throughout this section we define  $f_{\text{new}} := f(\mathbf{x} + \epsilon(\mathbf{x}' - \mathbf{x}))$  and  $f := f(\mathbf{x})$ . Furthermore, we define  $\mathcal{D} = \max_{i \in [n]} Df_i^\delta(\mathbf{x}, \mathbf{x}')$ . The following lemma, which is proved in Appendix B, gives a relationship between  $f$  and  $f_{\text{new}}$ .

**Lemma 11** *In every iteration of Algorithm 1 we have:*

$$f_{\text{new}} - f \leq \epsilon(\mathcal{D} - f) + \epsilon^2(1 - \mathcal{D}). \quad (11)$$

In the next lemma we prove that, if we are not in a  $\delta$ -stationary point, then we have a bound on the amount of progress made in each iteration. We use this in order to bound the number of iterations needed before we reach a point  $\mathbf{x}$  where  $f(\mathbf{x}) \leq 0.5 + \delta$ .

**Lemma 12** *Fix  $\epsilon = \frac{\delta}{\delta+2}$ , where  $0 < \delta \leq 0.5$ . Either  $\mathbf{x}$  is a  $\delta$ -stationary point or:*

$$f_{\text{new}} \leq \left( 1 - \left( \frac{\delta}{\delta+2} \right)^2 \right) f. \quad (12)$$

*Proof* Recall that by Lemma 11 the gain in every iteration of the steepest descent is

$$f_{new} - f \leq \epsilon(\mathcal{D} - f) + \epsilon^2(1 - \mathcal{D}). \quad (13)$$

We consider the following two cases:

- a)  $\mathcal{D} - f > -\delta$ . Then, by definition, we are in a  $\delta$ -stationary point.
- b)  $\mathcal{D} - f \leq -\delta$ . We have set  $\epsilon = \frac{\delta}{\delta+2}$ . If we solve for  $\delta$  we get that  $\delta = \frac{2\epsilon}{1-\epsilon}$ . Since  $\mathcal{D} - f \leq -\delta$ , we have that  $(\mathcal{D} - f)(1 - \epsilon) \leq -2\epsilon$ . Thus we have:

$$\begin{aligned} (\mathcal{D} - f)(\epsilon - 1) &\geq 2\epsilon \\ 0 &\geq (\mathcal{D} - f)(1 - \epsilon) + 2\epsilon \\ 0 &\geq (\mathcal{D} - f) + \epsilon(2 - \mathcal{D} + f) \\ -\epsilon f - \epsilon &\geq (\mathcal{D} - f) + \epsilon(1 - \mathcal{D}) \quad (\epsilon \geq 0) \\ -\epsilon^2 f - \epsilon^2 &\geq \epsilon(\mathcal{D} - f) + \epsilon^2(1 - \mathcal{D}). \end{aligned}$$

Thus, since  $\epsilon^2 \geq 0$  we get:

$$\begin{aligned} -\epsilon^2 f &\geq \epsilon(\mathcal{D} - f) + \epsilon^2(1 - \mathcal{D}) \\ &\geq f_{new} - f \end{aligned} \quad \text{According to (13).}$$

Thus we have shown that:

$$\begin{aligned} f_{new} - f &\leq -\epsilon^2 f \\ f_{new} &\leq (1 - \epsilon^2)f. \end{aligned}$$

Finally, using the fact that  $\epsilon = \frac{\delta}{\delta+2}$ , we get that

$$f_{new} \leq \left(1 - \left(\frac{\delta}{\delta+2}\right)^2\right) f.$$

□

So, when the algorithm has not reached yet a  $\delta$ -stationary point, there is a decrease on the value of  $f$  that is at least as large as the bound specified in (12) in every iteration of the gradient descent procedure. In the following lemma we prove that after  $O(\frac{1}{\delta^2})$  iterations of the steepest descent procedure the algorithm finds a point  $\mathbf{x}$  where  $f(\mathbf{x}) \leq 0.5 + \delta$ .

**Lemma 13** *After  $O(\frac{1}{\delta^2})$  iterations of the steepest descent procedure the algorithm finds a point  $\mathbf{x}$  where  $f(\mathbf{x}) \leq 0.5 + \delta$ .*

*Proof* Let  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k$  be the sequence of strategy profiles that are considered by Algorithm 1. Since the algorithm terminates as soon as it finds a  $(0.5 + \delta)$ -NE, we have  $f(\mathbf{x}_i) > 0.5 + \delta$  for every  $i < k$ . Therefore, for each  $i < k$  we can apply Lemma 10 to argue that  $\mathbf{x}_i$  is not a  $\delta$ -stationary point, which then allows us to apply Lemma 12 to obtain:

$$f(\mathbf{x}_{i+1}) \leq \left(1 - \left(\frac{\delta}{\delta+2}\right)^2\right) f(\mathbf{x}_i).$$

So, the amount of progress made by the algorithm in iteration  $i$  is:

$$\begin{aligned} f(\mathbf{x}_i) - f(\mathbf{x}_{i+1}) &\geq f(\mathbf{x}_i) - \left(1 - \left(\frac{\delta}{\delta+2}\right)^2\right) f(\mathbf{x}_i) \\ &= \left(\frac{\delta}{\delta+2}\right)^2 f(\mathbf{x}_i) \\ &\geq \left(\frac{\delta}{\delta+2}\right)^2 \cdot 0.5. \end{aligned}$$

Thus, each iteration of the algorithm decreases the regret by at least  $\left(\frac{\delta}{\delta+2}\right)^2 \cdot 0.5$ . The algorithm starts at a point  $\mathbf{x}_1$  with  $f(\mathbf{x}_1) \leq 1$ , and terminates when it reaches a point  $\mathbf{x}_k$  with  $f(\mathbf{x}_k) \leq 0.5 + \delta$ . Thus the total amount of progress made over all iterations of the algorithm can be at most  $1 - (0.5 + \delta)$ . Therefore, the number of iterations used by the algorithm can be at most:

$$\begin{aligned} \frac{1 - (0.5 + \delta)}{\left(\frac{\delta}{\delta+2}\right)^2 \cdot 0.5} &\leq \frac{1 - 0.5}{\left(\frac{\delta}{\delta+2}\right)^2 \cdot 0.5} \\ &= \frac{(\delta+2)^2}{\delta^2} = \frac{\delta^2}{\delta^2} + \frac{4\delta}{\delta^2} + \frac{4}{\delta^2}. \end{aligned}$$

Since  $\delta < 1$ , we have that the algorithm terminates after at most  $O(\frac{1}{\delta^2})$  iterations.  $\square$

Lemma 13 implies that after polynomially many iterations the algorithm finds a point such that  $f(\mathbf{x}) \leq 0.5 + \delta$ , and by definition such a point is a  $(0.5 + \delta)$ -NE. Thus we have completed the proof of Theorem 7.

## 7 Application: Two-player Bayesian games

In this section, we define two-player Bayesian games, and show how our algorithm can be applied in order to efficiently find a  $(0.5 + \delta)$ -Bayesian Nash equilibrium. A two-player Bayesian game is played between a *row* player and a *column* player. Each player has a set of possible *types*, and at the start of the game, each player is assigned a type by drawing from a known joint probability distribution. Each player learns his type, but not the type of his opponent. Our task is to find an approximate Bayesian Nash equilibrium (BNE).

We show that this can be reduced to the problem of finding an  $\epsilon$ -NE in a polymatrix game, and therefore our algorithm can be used to efficiently find a  $(0.5 + \delta)$ -BNE of a two-player Bayesian game. This section is split into two parts. In the first part we formally define two-player Bayesian games, and approximate Bayesian Nash equilibria. In the second part, we give the reduction from two-player Bayesian games to polymatrix games.

### 7.1 Definitions

**Payoff matrices.** We will use  $k_1$  to denote the number of pure strategies of the row player and  $k_2$  to denote the number of pure strategies of the column player. Furthermore, we will use  $m$  to denote the number of types of the row player, and  $n$  to denote the number of types of the column player.

For each pair of types  $i \in [m]$  and  $j \in [n]$ , there is a  $k_1 \times k_2$  bimatrix game  $(R, C)_{ij} := (R_{ij}, C_{ij})$  that is played when the row player has type  $i$  and the column player has type  $j$ . We assume that all payoffs in every matrix  $R_{ij}$  and every matrix  $C_{ij}$  lie in the range  $[0, 1]$ .

**Types.** The distribution over types is specified by a joint probability distribution: for each pair of types  $i \in [m]$  and  $j \in [n]$ , the probability that the row player is assigned type  $i$  and the column player is assigned type  $j$  is given by  $p_{ij}$ . Obviously, we have that:

$$\sum_{i=1}^m \sum_{j=1}^n p_{ij} = 1.$$

We also define some useful shorthands: for all  $i \in [m]$  we denote by  $p_i^R$  ( $p_j^C$ ) the probability that row (column) player has type  $i \in [m]$  ( $j \in [n]$ ). Formally:

$$\begin{aligned} p_i^R &= \sum_{j=1}^n p_{ij} && \text{for all } i \in [m], \\ p_j^C &= \sum_{i=1}^m p_{ij} && \text{for all } j \in [n]. \end{aligned}$$

Note that  $\sum_{i=1}^m p_i^R = \sum_{j=1}^n p_j^C = 1$ . Furthermore, we denote by  $p_i^R(j)$  the conditional probability that type  $j \in [n]$  will be chosen for column player given that type  $i$  is chosen for row player. Similarly, we define  $p_j^C(i)$  for the column player. Formally:

$$\begin{aligned} p_i^R(j) &= \frac{p_{ij}}{p_i^R} && \text{for all } i \in [m] \\ p_j^C(i) &= \frac{p_{ij}}{p_j^C} && \text{for all } j \in [n]. \end{aligned}$$

We can see that for given type  $t = (i, j)$  we have that  $p_{ij} = p_i^R \cdot p_i^R(j) = p_j^C \cdot p_j^C(i)$ .

**Strategies.** In order to play a Bayesian game, each player must specify a strategy for each of their types. Thus, a strategy profile is a pair  $(\mathbf{x}, \mathbf{y})$ , where  $\mathbf{x} = (x_1, x_2, \dots, x_m)$  such that each  $x_i \in \Delta_{k_1}$ , and where  $\mathbf{y} = (y_1, y_2, \dots, y_n)$  such that each  $y_j \in \Delta_{k_2}$ . This means that, when the row player gets type  $i \in [m]$  and the column player gets type  $j \in [n]$ , then the game  $(R_{ij}, C_{ij})$  will

be played, and the row player will use strategy  $x_i$  while the column player will use strategy  $y_j$ .

Given a strategy profile  $(\mathbf{x}, \mathbf{y})$ , we can define the expected payoff to both players (recall that the players are not told their opponent's type).

**Definition 14 (Expected payoff)** Given a strategy profile  $(\mathbf{x}, \mathbf{y})$  and a type  $t = (i, j)$ , the expected payoff for the row player is given by:

$$\begin{aligned} u_R(x_i, \mathbf{y}) &= \sum_{j=1}^n p_i^R(j) \cdot x_i^T R_{ij} y_j \\ &= x_i^T \sum_{j=1}^n p_i^R(j) \cdot R_{ij} y_j \end{aligned}$$

Similarly, for the column player the expected payoff is:

$$u_C(\mathbf{x}, y_j) = y_j^T \sum_{i=1}^m p_j^C(i) \cdot C_{ij}^T x_i.$$

**Rescaling.** Before we define approximate equilibria for two-player Bayesian games, we first rescale the payoffs. Much like for polymatrix games, rescaling is needed to ensure that an  $\epsilon$ -approximate equilibrium has a consistent meaning. Our rescaling will ensure that, for every possible pair of types, both player's expected payoff uses the entire range  $[0, 1]$ .

For each type  $i$  of the row player, we use  $U_R^i$  to denote the maximum expected payoff for the row player when he has type  $i$ , and we use  $L_R^i$  to denote the minimum expected payoff for the row player when he has type  $i$ . Formally, these are defined to be:

$$\begin{aligned} U_R^i &= \max_{a \in [k_1]} \sum_{j=1}^n \max_{b \in [k_2]} (p_i^R(j) \cdot R_{ij})_{a,b}, \\ L_R^i &= \min_{a \in [k_1]} \sum_{j=1}^n \min_{b \in [k_2]} (p_i^R(j) \cdot R_{ij})_{a,b}. \end{aligned}$$

Then we apply the transformation  $T_R^i(\cdot)$  to every element  $z$  of  $R_{ij}$ , for all types  $j$  of the column player, where:

$$T_R^i(z) := \frac{1}{U_R^i - L_R^i} \cdot \left( z - \frac{L_R^i}{n} \right). \quad (14)$$

Similarly, we transform all payoff matrices for the column player using

$$T_C^j(z) := \frac{1}{U_C^j - L_C^j} \cdot \left( z - \frac{L_C^j}{m} \right), \quad (15)$$

where  $U_C^j$  and  $L_C^j$  are defined symmetrically. Note that, after this transformation has been applied, both player's expected payoffs lie in the range  $[0, 1]$ .



Moreover, the full range is used: there exists a strategy for the column player against which one of the row player's strategies has expected payoff 1, and there exists a strategy for the column player against which one of the row player's strategies has expected payoff 0. From now on we will assume that the payoff matrices have been rescaled in this way.

We can now define approximate Bayesian Nash equilibria for a two-player Bayesian game.

**Definition 15 (Approximate Bayes Nash Equilibrium ( $\epsilon$ -BNE))** Let  $(\mathbf{x}, \mathbf{y})$  be a strategy profile. The profile  $(\mathbf{x}, \mathbf{y})$  is an  $\epsilon$ -BNE iff the following conditions hold:

$$u_R(x_i, \mathbf{y}) \geq u_R(x'_i, \mathbf{y}) - \epsilon \quad \text{for all } x'_i \in \Delta^{k_1} \quad \text{for all } i \in [m], \quad (16)$$

$$u_C(\mathbf{x}, y_j) \geq u_C(\mathbf{x}, y'_j) - \epsilon \quad \text{for all } y'_j \in \Delta^{k_2} \quad \text{for all } j \in [n]. \quad (17)$$

## 7.2 The reduction

In this section we reduce in polynomial time the problem of computing an  $\epsilon$ -BNE for a two-player Bayesian game  $\mathcal{B}$  to the problem of computing an  $\epsilon$ -NE of a polymatrix game  $\mathcal{P}(\mathcal{B})$ . We describe the construction of  $\mathcal{P}(\mathcal{B})$  and prove that every  $\epsilon$ -NE for  $\mathcal{P}(\mathcal{B})$  maps to an  $\epsilon$ -BNE of  $\mathcal{B}$ .

**Construction.** Let  $\mathcal{B}$  be a two-player Bayesian game where the row player has  $m$  types and  $k_1$  pure strategies and the column player has  $n$  types and  $k_2$  pure strategies. We will construct a polymatrix game  $\mathcal{P}(\mathcal{B})$  as follows.

The game has  $m + n$  players. We partition the set of players  $[m + n]$  into two sets: the set  $K = \{1, 2, \dots, m\}$  will represent the types of the row player in  $\mathcal{B}$ , while the set  $L = \{m + 1, m + 2, \dots, m + n\}$  will represent the types of the column player in  $\mathcal{B}$ . The underlying graph that shows the interactions between the players is a complete bipartite graph  $G = (K \cup L, E)$ , where every player in  $K$  (respectively  $L$ ) plays a bimatrix game with every player in  $L$  (respectively  $K$ ). The bimatrix game played between vertices  $v_i \in K$  and  $v_j \in L$  is defined to be  $(R_{ij}^*, C_{ij}^*)$ , where:

$$R_{ij}^* := p_i^R(j) \cdot R_{ij} \quad (18)$$

$$C_{ij}^* := p_j^C(i) \cdot C_{ij} \quad (19)$$

for all  $i \in [m]$  and  $j \in [n]$ .

Observe that, for each player  $i$  in the  $K$ , the matrices  $R_{ij}^*$  all have the same number of rows, and for each player  $j \in L$ , the matrices  $C_{ij}^*$  all have the same number of columns. Thus,  $\mathcal{P}(\mathcal{B})$  is a valid polymatrix game. Moreover, we clearly have that  $\mathcal{P}(\mathcal{B})$  has the same size as the original game  $\mathcal{B}$ . Note that, since we have assumed that the Bayesian game has been rescaled, we have that for every player in  $\mathcal{P}(\mathcal{B})$  the minimum (maximum) payoff achievable under pure strategy profiles is 0 (1), so no further scaling is needed in order to apply our algorithm.

We can now prove that every  $\epsilon$ -NE of the polymatrix game is also an  $\epsilon$ -BNE of the original two-player Bayesian game, which is the main result of this section.

**Theorem 16** *Every  $\epsilon$ -NE of  $\mathcal{P}(\mathcal{B})$  is a  $\epsilon$ -BNE for  $\mathcal{B}$ .*

*Proof* Let  $\mathbf{z} = (x_1, \dots, x_m, y_1, \dots, y_n)$  be an  $\epsilon$ -NE for  $\mathcal{P}(\mathcal{B})$ . This means that no player can gain more than  $\epsilon$  by unilaterally changing his strategy. We define the strategy profile  $(\mathbf{x}, \mathbf{y})$  for  $\mathcal{B}$  where  $\mathbf{x} = (x_1, \dots, x_m)$  and  $\mathbf{y} = (y_1, \dots, y_n)$ , and we will show that  $(\mathbf{x}, \mathbf{y})$  is an  $\epsilon$ -BNE for  $\mathcal{B}$ .

Let  $i \in K$  be a player. Since,  $\mathbf{z}$  is an  $\epsilon$ -NE of  $\mathcal{P}(\mathcal{B})$ , we have:

$$u_i(x_i, \mathbf{z}) \geq u_i(x'_i, \mathbf{z}) - \epsilon \quad \text{for all } x'_i \in \Delta^{k_1}.$$

By construction, we can see that player  $i$  only interacts with the players from  $L$ . Hence his payoff can be written as:

$$u_i(x_i, \mathbf{z}) = x_i^T \sum_{j=1}^n R_{ij}^* y_j = u^R(x_i, \mathbf{y})$$

and since we are in an  $\epsilon$ -NE, we have:

$$u^R(x_i, \mathbf{y}) \geq u^R(x'_i, \mathbf{y}) - \epsilon \quad \text{for all } x'_i \in \Delta^{k_1}. \quad (20)$$

This is true for all  $i \in K$ , thus it is true for all  $i \in [m]$ .

Similarly, every player  $j \in L$  interacts only with players from  $K$ , thus:

$$u^C(\mathbf{x}, y_j) = y_j^T \sum_{i=1}^m (C_{ij}^*)^T x_i.$$

and since we are in an  $\epsilon$ -NE we have:

$$u^C(\mathbf{x}, y_j) \geq u^C(\mathbf{x}, y'_j) - \epsilon \quad \text{for all } y'_j \in \Delta^{k_2} \quad (21)$$

and this is true for all  $j \in K$ , thus it is true for all  $j \in [n]$ .

Combining now the fact that Equation (20) is true for all  $i \in [m]$  and that Equation (21) is true for all  $j \in [n]$ , it is easy to see that the strategy profile  $(\mathbf{x}, \mathbf{y})$  is an  $\epsilon$ -BNE for  $\mathcal{B}$ .  $\square$

Applying Algorithm 1 to  $\mathcal{P}(\mathcal{B})$  thus gives us the following.

**Theorem 17** *A  $(0.5 + \delta)$ -Bayesian Nash equilibrium of a two-player Bayesian game  $\mathcal{B}$  can be found in time polynomial in the input size of  $\mathcal{B}$  and  $1/\delta$ .*

## 8 Conclusions and open questions

We have presented a polynomial-time algorithm that finds a  $(0.5 + \delta)$ -Nash equilibrium of a polymatrix game for any  $\delta > 0$ . Though we do not have examples that show that the approximation guarantee is tight for our algorithm, we do not see an obvious approach to prove a better guarantee. The initial choice of strategy profile affects our algorithm, and it is conceivable that one may be able to start the algorithm from an efficiently computable profile with certain properties that allow a better approximation guarantee. One natural special case is when there is a constant number of players, which may allow one to derive new strategy profiles from a stationary point as done by Tsaknakis and Spirakis [24]. It may also be possible to develop new techniques when the number of pure strategies available to the players is constant, or when the structure of the graph is restricted in some way. For example, in the games arising from two-player Bayesian games, the graph is always bipartite.

This paper has considered  $\epsilon$ -Nash equilibria, which are the most well-studied type of approximate equilibria. However,  $\epsilon$ -Nash equilibria have a drawback: since they only require that the expected payoff is within  $\epsilon$  of a pure best response, it is possible that a player could be required to place probability on a strategy that is arbitrarily far from being a best response. An alternative, stronger, notion is an  $\epsilon$ -well supported approximate Nash equilibrium ( $\epsilon$ -WSNE). It requires that players only place probability on strategies that have payoff within  $\epsilon$  of a pure best response. Every  $\epsilon$ -WSNE is an  $\epsilon$ -Nash, but the converse is not true. For bimatrix games, the best-known additive approximation that is achievable in polynomial time gives a  $(\frac{2}{3} - 0.0047)$ -WSNE [15]. It builds on the algorithm given by Kontogiannis and Spirakis that achieves a  $\frac{2}{3}$ -WSNE in polynomial time [21]. Recently a polynomial-time algorithm with a better approximation guarantee have been given for *symmetric* bimatrix games [7]. Note, it has been shown that there is a PTAS for finding  $\epsilon$ -WSNE of bimatrix games if and only if there is a PTAS for  $\epsilon$ -Nash [9, 5]. For  $n$ -player games with  $n > 2$  there has been very little work on developing algorithms for finding  $\epsilon$ -WSNE. This is a very interesting direction, both in general and when  $n > 2$  is a constant.

**Acknowledgements** We thank Aviad Rubinstein for alerting us to the two-player Bayesian games application, and Haralampos Tsaknakis for feedback on earlier versions of this paper.

## References

1. Babichenko, Y., Barman, S., Peretz, R.: Simple approximate equilibria in large games. In: EC, pp. 753–770 (2014)
2. Bosse, H., Byrka, J., Markakis, E.: New algorithms for approximate Nash equilibria in bimatrix games. Theoretical Computer Science **411**(1), 164–173 (2010)
3. Briest, P., Goldberg, P., Röglin, H.: Approximate equilibria in games with few players. CoRR **abs/0804.4524** (2008)

4. Cai, Y., Daskalakis, C.: On minmax theorems for multiplayer games. In: SODA, pp. 217–234 (2011)
5. Chen, X., Deng, X., Teng, S.H.: Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM* **56**(3), 14:1–14:57 (2009)
6. Chen, X., Paparas, D., Yannakakis, M.: The complexity of non-monotone markets. In: STOC, pp. 181–190 (2013)
7. Czumaj, A., Fasoulakis, M., Jurdziński, M.: Approximate well-supported Nash equilibria in symmetric bimatrix games. In: Proc. of SAGT (2014). To appear.
8. Daskalakis, C.: On the complexity of approximating a Nash equilibrium. *ACM Transactions on Algorithms* **9**(3), 23 (2013)
9. Daskalakis, C., Goldberg, P.W., Papadimitriou, C.H.: The complexity of computing a Nash equilibrium. *SIAM Journal on Computing* **39**(1), 195–259 (2009)
10. Daskalakis, C., Mehta, A., Papadimitriou, C.H.: Progress in approximate Nash equilibria. In: Proceedings of ACM-EC, pp. 355–358 (2007)
11. Daskalakis, C., Mehta, A., Papadimitriou, C.H.: A note on approximate Nash equilibria. *Theoretical Computer Science* **410**(17), 1581–1588 (2009)
12. Daskalakis, C., Papadimitriou, C.H.: Approximate Nash equilibria in anonymous games. *Journal of Economic Theory* (2014). DOI <http://dx.doi.org/10.1016/j.jet.2014.02.002>. To appear. <http://dx.doi.org/10.1016/j.jet.2014.02.002>
13. Deligkas, A., Fearnley, J., Savani, R., Spirakis, P.: Computing approximate Nash equilibria in polymatrix games. In: Proc. of WINE (2014). To appear.
14. Etessami, K., Yannakakis, M.: On the complexity of nash equilibria and other fixed points. *SIAM J. Comput.* **39**(6), 2531–2597 (2010)
15. Fearnley, J., Goldberg, P.W., Savani, R., Sørensen, T.B.: Approximate well-supported Nash equilibria below two-thirds. In: SAGT, pp. 108–119 (2012)
16. Feige, U., Talgam-Cohen, I.: A direct reduction from  $k$ -player to 2-player approximate Nash equilibrium. In: SAGT, pp. 138–149 (2010)
17. Govindan, S., Wilson, R.: Computing Nash equilibria by iterated polymatrix approximation. *Journal of Economic Dynamics and Control* **28**(7), 1229–1241 (2004)
18. Govindan, S., Wilson, R.: A decomposition algorithm for  $n$ -player games. *Economic Theory* **42**(1), 97–117 (2010). DOI 10.1007/s00199-009-0434-4. URL <http://dx.doi.org/10.1007/s00199-009-0434-4>
19. Hémon, S., de Rougemont, M., Santha, M.: Approximate Nash equilibria for multiplayer games. In: SAGT, pp. 267–278 (2008)
20. Howson Joseph T., J.: Equilibria of polymatrix games. *Management Science* **18**(5), pp. 312–318 (1972)
21. Kontogiannis, S.C., Spirakis, P.G.: Well supported approximate equilibria in bimatrix games. *Algorithmica* **57**(4), 653–667 (2010)
22. Lipton, R.J., Markakis, E., Mehta, A.: Playing large games using simple strategies. In: EC, pp. 36–41 (2003)
23. Rubinstein, A.: Inapproximability of Nash equilibrium. *CoRR* **abs/1405.3322** (2014)
24. Tsaknakis, H., Spirakis, P.G.: An optimization approach for approximate Nash equilibria. *Internet Mathematics* **5**(4), 365–382 (2008)

### A Proof of Lemma 3

Before we begin with the proof, we introduce the following notation. For a player  $i \in [n]$ , given a strategy profile  $\mathbf{x}$  and a subset of  $i$ 's pure strategies  $S \subseteq [m_i]$ , we use  $M_i(\mathbf{x}, S)$  for taking the maximum of the payoffs of  $i$  when the others play according to  $\mathbf{x}$ , and player  $i$  is restricted to pick elements from  $S$ :

$$M_i(\mathbf{x}, S) := \max_S v_i(\mathbf{x}).$$

In order to find the gradient, we have to calculate the variation of  $f_i$  along the direction  $\mathbf{x}' - \mathbf{x}$ , by evaluating  $f(\bar{\mathbf{x}})$  for points  $\bar{\mathbf{x}}$  of the form

$$\bar{\mathbf{x}} := \mathbf{x} + \epsilon(\mathbf{x}' - \mathbf{x}) = (1 - \epsilon) \cdot \mathbf{x} + \epsilon \cdot \mathbf{x}'.$$

Recall from (4), that for  $\bar{\mathbf{x}} \in \Delta$  we have that  $f_i(\bar{\mathbf{x}}) := u_i^*(\bar{\mathbf{x}}) - u_i(\bar{\mathbf{x}})$ . In order to rewrite  $u_i^*(\bar{\mathbf{x}})$  we introduce notation  $\Lambda_i(\mathbf{x}, \mathbf{x}', \epsilon)$  as follows.

**Definition 18** Given  $(\mathbf{x}, \mathbf{x}', \epsilon)$  and  $S = \text{Br}_i(\mathbf{x})$  we define  $\Lambda_i(\mathbf{x}, \mathbf{x}', \epsilon)$  as:

$$\Lambda_i(\mathbf{x}, \mathbf{x}', \epsilon) := \max \left\{ 0, \max_{k \in S} \{(v_i(\bar{\mathbf{x}}))_k\} - \max_{l \in S} \{(v_i(\bar{\mathbf{x}}))_l\} \right\}. \quad (22)$$

In the following technical lemma we provide an expression for  $u_i^*(\bar{\mathbf{x}})$ . In order to rewrite  $u_i^*(\bar{\mathbf{x}})$ , we use the following simple observation. Consider a multiset of numbers  $\{a_1, \dots, a_n\}$ , and the index sets  $S \subseteq [n]$  and  $\bar{S} = [n] \setminus S$ . We have the following identity:

$$\max\{a_1, \dots, a_n\} \equiv \max_{j \in S} \{a_j\} + \max \left\{ 0, \max_{k \in \bar{S}} \{a_k\} - \max_{j \in S} \{a_j\} \right\}. \quad (23)$$

In the following lemma, we use this identity with  $S = \text{Br}_i(\mathbf{x})$  to rewrite  $u_i^*(\bar{\mathbf{x}})$ .

**Lemma 19** Given profiles  $\mathbf{x}$  and  $\mathbf{x}'$  in  $\Delta$  and a player  $i \in [n]$ , let  $S = \text{Br}_i(\mathbf{x})$ . We have:

$$u_i^*((1 - \epsilon) \cdot \mathbf{x} + \epsilon \cdot \mathbf{x}') = (1 - \epsilon) \cdot M_i(\mathbf{x}, S) + \epsilon \cdot M_i(\mathbf{x}', S) + \Lambda_i(\mathbf{x}, \mathbf{x}', \epsilon). \quad (24)$$

*Proof*

$$\begin{aligned} u_i^*(\bar{\mathbf{x}}) &= u_i^*((1 - \epsilon) \cdot \mathbf{x} + \epsilon \cdot \mathbf{x}') \\ &= \max_{k \in [m_i]} \{ (v_i(\mathbf{x} + \epsilon(\mathbf{x}' - \mathbf{x})))_k \} && \text{By (3)} \\ &= \max_{k \in S} \{ (v_i(\mathbf{x} + \epsilon(\mathbf{x}' - \mathbf{x})))_k \} + \Lambda_i(\mathbf{x}, \mathbf{x}', \epsilon) && \text{By (23) and (22)} \\ &= \max_{k \in S} \{ ((1 - \epsilon) \cdot v_i(\mathbf{x}) + \epsilon \cdot v_i(\mathbf{x}'))_k \} + \Lambda_i(\mathbf{x}, \mathbf{x}', \epsilon). \end{aligned}$$

Since  $S = \text{Br}_i(\mathbf{x})$ , we know that for all  $k \in S$  we have that  $(v_i(\mathbf{x}))_k$  are equal, so we have the following:

$$\begin{aligned} \max_{k \in S} \{ ((1 - \epsilon) \cdot v_i(\mathbf{x}) + \epsilon \cdot v_i(\mathbf{x}'))_k \} &= \max_{k \in S} \{ ((1 - \epsilon) \cdot v_i(\mathbf{x}))_k \} + \max_{k \in S} \{ (\epsilon \cdot v_i(\mathbf{x}'))_k \} \\ &= (1 - \epsilon) \cdot M_i(\mathbf{x}, S) + \epsilon \cdot M_i(\mathbf{x}', S) \end{aligned}$$

and we get the claimed result.  $\square$

We will use the expression (24) for  $u_i^*(\bar{\mathbf{x}})$ , along with the following reformulation of  $u_i(\bar{\mathbf{x}})$ :

$$\begin{aligned} u_i(\bar{\mathbf{x}}) &= u_i(\mathbf{x} + \epsilon(\mathbf{x}' - \mathbf{x})) \\ &= u_i(x_i + \epsilon(x'_i - x_i), \mathbf{x} + \epsilon(\mathbf{x}' - \mathbf{x})) \\ &= u_i(x_i, \mathbf{x}) + \epsilon \cdot u_i(x_i, \mathbf{x}' - \mathbf{x}) + \epsilon \cdot u_i(x'_i - x_i, \mathbf{x}) + \epsilon^2 \cdot u_i(x'_i - x_i, \mathbf{x}' - \mathbf{x}) \\ &= u_i(\mathbf{x}) + \epsilon \cdot u_i(x_i, \mathbf{x}') - \epsilon \cdot u_i(x_i, \mathbf{x}) + \epsilon \cdot u_i(x'_i, \mathbf{x}) + \epsilon \cdot u_i(x_i, \mathbf{x}) - \epsilon^2 \cdot u_i(\mathbf{x}' - \mathbf{x}) \\ &= (1 - \epsilon) \cdot u_i(\mathbf{x}) + \epsilon(u_i(x_i, \mathbf{x}') + u_i(x'_i, \mathbf{x}) - u_i(\mathbf{x})) + \epsilon^2 \cdot u_i(\mathbf{x}' - \mathbf{x}). \end{aligned} \quad (25)$$

We now use these reformulations to prove the following lemma.

**Lemma 20** We have that  $f_i(\bar{\mathbf{x}}) - f(\mathbf{x})$  is equal to:

$$\epsilon(Df_i(\mathbf{x}, \mathbf{x}') - f(\mathbf{x})) + \Lambda_i(\mathbf{x}, \mathbf{x}', \epsilon) - \epsilon^2 u_i(\mathbf{x}' - \mathbf{x}) - (1 - \epsilon) \max_{j \in [n]} \{f_j(\mathbf{x}) - f_i(\mathbf{x})\}.$$

*Proof* Recall that  $S = \text{Br}_i(\mathbf{x})$ . For a given  $i \in [n]$ , using Lemma 19 and the reformulation for  $u_i(\bar{\mathbf{x}})$ , we have:

$$\begin{aligned} f_i(\bar{\mathbf{x}}) - f(\mathbf{x}) &= u_i^*(\bar{\mathbf{x}}) - u_i(\bar{\mathbf{x}}) - f(\mathbf{x}) \\ &= (1 - \epsilon) \cdot M_i(\mathbf{x}, S) + \epsilon \cdot M_i(\mathbf{x}', S) + \Lambda_i(\mathbf{x}, \mathbf{x}', \epsilon) \\ &\quad - (1 - \epsilon)u_i(\mathbf{x}) + \epsilon(-u_i(x_i, \mathbf{x}') - u_i(x'_i, \mathbf{x}) + u_i(\mathbf{x})) - \epsilon^2 u_i(\mathbf{x}' - \mathbf{x}) - f(\mathbf{x}). \end{aligned}$$

Recall from (4) that  $f_i(\mathbf{x}) = M_i(\mathbf{x}, S) - u_i(\mathbf{x})$ , so the formula above is equal to:

$$\epsilon(M_i(\mathbf{x}', S) - u_i(x_i, \mathbf{x}') - u_i(x'_i, \mathbf{x}) + u_i(\mathbf{x})) + \Lambda_i(\mathbf{x}, \mathbf{x}', \epsilon) - \epsilon^2 u_i(\mathbf{x}' - \mathbf{x}) + (1 - \epsilon)f_i(\mathbf{x}) - f(\mathbf{x}).$$

Using now (6) for  $Df_i(\mathbf{x}, \mathbf{x}')$ , the above formula becomes:

$$\begin{aligned} \epsilon \cdot Df_i(\mathbf{x}, \mathbf{x}') + \Lambda_i(\mathbf{x}, \mathbf{x}', \epsilon) - \epsilon^2 u_i(\mathbf{x}' - \mathbf{x}) + (1 - \epsilon)f_i(\mathbf{x}) - f(\mathbf{x}) &= \\ \epsilon \cdot Df_i(\mathbf{x}, \mathbf{x}') + \Lambda_i(\mathbf{x}, \mathbf{x}', \epsilon) - \epsilon^2 u_i(\mathbf{x}' - \mathbf{x}) + (1 - \epsilon)f_i(\mathbf{x}) - (1 - \epsilon)f(\mathbf{x}) - \epsilon f(\mathbf{x}) &= \\ \epsilon(Df_i(\mathbf{x}, \mathbf{x}') - f(\mathbf{x})) + \Lambda_i(\mathbf{x}, \mathbf{x}', \epsilon) - \epsilon^2 u_i(\mathbf{x}' - \mathbf{x}) - (1 - \epsilon)(f(\mathbf{x}) - f_i(\mathbf{x})). \end{aligned}$$

Recall now that  $f(\mathbf{x}) = \max_{j \in [n]} f_j(\mathbf{x})$ . Thus the term  $f(\mathbf{x}) - f_i(\mathbf{x})$  can be written as  $\max_{j \in [n]} \{f_j(\mathbf{x}) - f_i(\mathbf{x})\}$ . So, the expression above is equivalent to

$$\epsilon(Df_i(\mathbf{x}, \mathbf{x}') - f(\mathbf{x})) + \Lambda_i(\mathbf{x}, \mathbf{x}', \epsilon) - \epsilon^2 u_i(\mathbf{x}' - \mathbf{x}) - (1 - \epsilon) \max_{j \in [n]} \{f_j(\mathbf{x}) - f_i(\mathbf{x})\}.$$

□

Now we are ready to prove Lemma 3. Recall from definition 1 for the gradient that

$$\begin{aligned} Df(\mathbf{x}, \mathbf{x}') &= \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} (f((1 - \epsilon) \cdot \mathbf{x} + \epsilon \cdot \mathbf{x}') - f(\mathbf{x})) \\ &= \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} (f(\bar{\mathbf{x}}) - f(\mathbf{x})) \\ &= \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} \left( \max_{i \in [n]} f_i(\bar{\mathbf{x}}) - f(\mathbf{x}) \right) \\ &= \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} \left( \max_{i \in [n]} (f_i(\bar{\mathbf{x}}) - f(\mathbf{x})) \right) \\ &= \max_{i \in [n]} \left( \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} (f_i(\bar{\mathbf{x}}) - f(\mathbf{x})) \right). \end{aligned} \tag{26}$$

We will now use lemma 20 to study the limit  $\lim_{\epsilon \rightarrow 0} (f_i(\bar{\mathbf{x}}) - f(\mathbf{x}))$  for all  $i \in [n]$ . Firstly, we deal with  $\Lambda_i(\mathbf{x}, \mathbf{x}', \epsilon)$ . It is easy to see that  $\lim_{\epsilon \rightarrow 0} (\mathbf{x} + \epsilon(\mathbf{x}' - \mathbf{x})) = \mathbf{x}$ . Then, when  $S = \text{Br}_i(\mathbf{x})$  we have that

$$\lim_{\epsilon \rightarrow 0} \left( \max_{k \in S} \{v_i(\bar{\mathbf{x}})\}_k - \max_{l \in S} \{v_i(\bar{\mathbf{x}})\}_l \right) < 0.$$

This is true from the definition of pure best response strategies. So, from equation (22) for  $\Lambda_i(\mathbf{x}, \mathbf{x}', \epsilon)$  it is true that  $\lim_{\epsilon \rightarrow 0} \Lambda_i(\mathbf{x}, \mathbf{x}', \epsilon) = 0$ .

Furthermore, the term  $\epsilon^2 \cdot u_i(\mathbf{x}' - \mathbf{x})$  when is divided by  $\epsilon$  equals to  $\epsilon \cdot u_i(\mathbf{x}' - \mathbf{x})$ , thus  $\lim_{\epsilon \rightarrow 0} (\epsilon \cdot u_i(\mathbf{x}' - \mathbf{x})) = 0$ .

Moreover, the term

$$\lim_{\epsilon \rightarrow 0} \left( -\frac{1 - \epsilon}{\epsilon} \cdot \max_{j \in [n]} \{f_j(\mathbf{x}) - f_i(\mathbf{x})\} \right)$$

is either 0 when  $f_i(\mathbf{x}) = f(\mathbf{x})$ , i.e player  $i$  has the maximum regret and  $\max_{j \in [n]} \{f_j(\mathbf{x}) - f_i(\mathbf{x})\} = 0$ , or  $-\infty$  otherwise, because  $\max_{j \in [n]} \{f_j(\mathbf{x}) - f_i(\mathbf{x})\} > 0$ .

To sum up, if  $f_i(\mathbf{x})$  achieves the maximum regret at point  $\mathbf{x}'$ , then the limit  $\lim_{\epsilon \rightarrow 0} (f_i(\bar{\mathbf{x}}) - f(\mathbf{x})) = Df_i(\mathbf{x}, \mathbf{x}') - f(\mathbf{x})$ , otherwise the limit equals  $-\infty$ .

From (26) for the gradient we want the maximum of these quantities, thus we have the claimed result.

## B Proof of Lemma 11

Throughout this proof,  $\mathbf{x}, \mathbf{x}', \bar{\mathbf{x}}$ , and  $\epsilon$  will be fixed as they are defined in Section 6. In order to prove this lemma, we must show a bound on:

$$f(\bar{\mathbf{x}}) - f(\mathbf{x}) = \max_{i \in [n]} f_i(\bar{\mathbf{x}}) - f(\mathbf{x}).$$

Before we start the analysis we need to redefine the term  $\Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon)$  in order to prove an analogous version of Lemma 19 when  $\delta$ -best responses are used.

**Definition 21** We define  $\Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon)$  as:

$$\Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon) := \max \left\{ 0, \max_{k \in \text{Br}_i^\delta(\mathbf{x})} \{(v_i(\bar{\mathbf{x}}))_k\} - \max_{l \in \text{Br}_i^\delta(\mathbf{x})} \{(v_i(\bar{\mathbf{x}}))_l\} \right\}. \quad (27)$$

We now use this definition to prove the following lemma.

**Lemma 22** *We have:*

$$u_i^*((1 - \epsilon) \cdot \mathbf{x} + \epsilon \cdot \mathbf{x}') \leq (1 - \epsilon) \max_{k \in \text{Br}_i^\delta(\mathbf{x})} (v_i(\mathbf{x}))_k + \epsilon \max_{k \in \text{Br}_i^\delta(\mathbf{x})} (v_i(\mathbf{x}'))_k + \Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon). \quad (28)$$

*Proof* We have:

$$\begin{aligned} u_i^*((1 - \epsilon) \cdot \mathbf{x} + \epsilon \cdot \mathbf{x}') &= \max_{k \in [m_i]} (v_i((1 - \epsilon) \cdot \mathbf{x} + \epsilon \cdot \mathbf{x}'))_k \\ &= \max_{k \in \text{Br}_i^\delta(\mathbf{x})} (v_i((1 - \epsilon) \cdot \mathbf{x} + \epsilon \cdot \mathbf{x}'))_k + \Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon) \quad \text{Using (23)} \\ &\leq (1 - \epsilon) \max_{k \in \text{Br}_i^\delta(\mathbf{x})} (v_i(\mathbf{x}))_k + \epsilon \max_{k \in \text{Br}_i^\delta(\mathbf{x})} (v_i(\mathbf{x}'))_k + \Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon). \end{aligned}$$

□

We will use the reformulation from Equation (25) for  $u_i(\bar{\mathbf{x}})$ :

$$u_i(\bar{\mathbf{x}}) = (1 - \epsilon) \cdot u_i(\mathbf{x}) + \epsilon(u_i(x_i, \mathbf{x}') + u_i(x'_i, \mathbf{x}) - u_i(\mathbf{x})) + \epsilon^2 \cdot u_i(\mathbf{x}' - \mathbf{x}). \quad (29)$$

The correctness of this was proved in Appendix A. Now we use all the these reformulations in order to prove the following lemma.

**Lemma 23** *We have that  $f_i(\bar{\mathbf{x}}) - f(\mathbf{x})$  is less than or equal to:*

$$\epsilon(Df_i^\delta(\mathbf{x}, \mathbf{x}') - f(\mathbf{x})) + \Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon) - \epsilon^2 u_i(\mathbf{x}' - \mathbf{x}) - (1 - \epsilon) \max_{j \in [n]} \{f_j - f_i\}. \quad (30)$$

*Proof* Recall that, by definition, we have that:

$$f_i(\bar{\mathbf{x}}) = u_i^*(\bar{\mathbf{x}}) - u_i(\bar{\mathbf{x}}).$$

Thus, we can apply Lemma 22 along with the reformulation given in Equation (29) for  $u_i(\bar{\mathbf{x}})$  to prove that  $f_i(\bar{\mathbf{x}}) - f(\mathbf{x})$  is less than or equal to:

$$(1 - \epsilon) \max_{k \in \text{Br}_i^\delta(\mathbf{x})} (v_i(\mathbf{x}))_k + \epsilon \max_{k \in \text{Br}_i^\delta(\mathbf{x})} (v_i(\mathbf{x}'))_k + \Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon) \\ - (1 - \epsilon)u_i(\mathbf{x}) + \epsilon(-u_i(x_i, \mathbf{x}') - u_i(x'_i, \mathbf{x}) + u_i(\mathbf{x})) - \epsilon^2 u_i(\mathbf{x}' - \mathbf{x}) - f(\mathbf{x}).$$

We can now use the fact that  $\max_{k \in \text{Br}_i^\delta(\mathbf{x})} (v_i(\mathbf{x}))_k - u_i(\mathbf{x}) = f_i(\mathbf{x})$  and the definition of  $Df_i^\delta(\mathbf{x}, \mathbf{x}')$  given in (8) to prove that the expression above is equivalent to:

$$\epsilon \cdot Df_i^\delta(\mathbf{x}, \mathbf{x}') + \Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon) - \epsilon^2 u_i(\mathbf{x}' - \mathbf{x}) + (1 - \epsilon)f_i(\mathbf{x}) - f(\mathbf{x}) \\ = \epsilon \cdot Df_i^\delta(\mathbf{x}, \mathbf{x}') + \Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon) - \epsilon^2 u_i(\mathbf{x}' - \mathbf{x}) + (1 - \epsilon)f_i(\mathbf{x}) - (1 - \epsilon)f(\mathbf{x}) - \epsilon f(\mathbf{x}) \\ = \epsilon(Df_i^\delta(\mathbf{x}, \mathbf{x}') - f(\mathbf{x})) + \Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon) - \epsilon^2 u_i(\mathbf{x}' - \mathbf{x}) - (1 - \epsilon)(f(\mathbf{x}) - f_i(\mathbf{x})) \\ = \epsilon(Df_i^\delta(\mathbf{x}, \mathbf{x}') - f(\mathbf{x})) + \Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon) - \epsilon^2 u_i(\mathbf{x}' - \mathbf{x}) - (1 - \epsilon) \max_{j \in [n]} \{f_j(\mathbf{x}) - f_i(\mathbf{x})\}.$$

This completes the proof.  $\square$

Having shown Lemma 23, we will now study each term of (30) and provide bounds for each of them. To begin with, it is easy to see that for all  $i \in [n]$  we have that  $\max_{j \in [n]} \{f_j(\mathbf{x}) - f_i(\mathbf{x})\} \geq 0$ , and since  $\epsilon < 1$ , we have that  $(1 - \epsilon) \max_{j \in [n]} \{f_j(\mathbf{x}) - f_i(\mathbf{x})\} \geq 0$ . Thus, Equation (30) is less than or equal to:

$$\epsilon(Df_i^\delta(\mathbf{x}, \mathbf{x}') - f(\mathbf{x})) + \Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon) - \epsilon^2 u_i(\mathbf{x}' - \mathbf{x}). \quad (31)$$

Next we consider the term  $\Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon)$ . In the following technical lemma we prove that  $\Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon) = 0$  for all  $i \in [n]$ .

**Lemma 24** *We have  $\Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon) = 0$  for all  $i \in [n]$ .*

*Proof* According to equation (27) for  $\Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon)$ , we have:

$$\Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon) = \max \left\{ 0, \max_{k \in \text{Br}_i^\delta(\mathbf{x})} \{(v_i(\bar{\mathbf{x}}))_k\} - \max_{l \in \text{Br}_i^\delta(\mathbf{x})} \{(v_i(\bar{\mathbf{x}}))_l\} \right\}.$$

We can rewrite this expression as follows. First define:

$$Z(\mathbf{x}, \mathbf{x}', \epsilon, k) = (v_i(\bar{\mathbf{x}}))_k - \max_{l \in \text{Br}_i^\delta(\mathbf{x})} \{(v_i(\bar{\mathbf{x}}))_l\}.$$

Then we have:

$$\Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon) = \max \left\{ 0, \max_{k \in \text{Br}_i^\delta(\mathbf{x})} \{Z(\mathbf{x}, \mathbf{x}', \epsilon, k)\} \right\}.$$

Our goal is to show that, for our chosen value of  $\epsilon$ , we have  $\Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon) = 0$ . For this to be the case, we must have that  $Z(\mathbf{x}, \mathbf{x}', \epsilon, k) \leq 0$  for all  $k \in \text{Br}_i^\delta(\mathbf{x})$ . In the rest of this proof, we will show that this is indeed the case.

By definition, we have that:

$$(v_i(\bar{\mathbf{x}}))_k = (v_i(\mathbf{x}) + \epsilon(v_i(\mathbf{x}') - v_i(\mathbf{x})))_k. \quad (32)$$



The term  $\max_{l \in \text{Br}_i^\delta(\mathbf{x})} \{(v_i(\bar{\mathbf{x}}))_l\}$  can be written as follows:

$$\begin{aligned} \max_{l \in \text{Br}_i^\delta(\mathbf{x})} \{(v_i((1-\epsilon)\mathbf{x} + \epsilon\mathbf{x}'))_l\} &\geq \max_{l \in \text{Br}_i^\delta(\mathbf{x})} \{(v_i((1-\epsilon)\mathbf{x}))_l\} \\ &= (1-\epsilon) \cdot \max_{l \in \text{Br}_i^\delta(\mathbf{x})} \{(v_i(\mathbf{x}))_l\} \\ &= \max_{l \in \text{Br}_i^\delta(\mathbf{x})} \{(v_i(\mathbf{x}))_l\} - \epsilon \cdot \max_{l \in \text{Br}_i^\delta(\mathbf{x})} \{(v_i(\mathbf{x}))_l\}. \end{aligned} \quad (33)$$

We now substitute these two bounds into the definition of  $Z(\mathbf{x}, \mathbf{x}', \epsilon, k)$ . We have:

$$Z(\mathbf{x}, \mathbf{x}', \epsilon, k) \leq v_i(\mathbf{x})_k - \max_{l \in \text{Br}_i^\delta(\mathbf{x})} \{(v_i(\mathbf{x}))_l\} + \epsilon \left( v_i(\mathbf{x}')_k - v_i(\mathbf{x})_k + \max_{l \in \text{Br}_i^\delta(\mathbf{x})} \{(v_i(\mathbf{x}))_l\} \right). \quad (34)$$

From the definition of  $\delta$ -best responses (Definition 4), we know that for all  $k \in \overline{\text{Br}_i^\delta(\mathbf{x})}$ :

$$v_i(\mathbf{x})_k - \max_{l \in \text{Br}_i^\delta(\mathbf{x})} \{(v_i(\mathbf{x}))_l\} < -\delta.$$

Furthermore, since we know that the maximum payoff for player  $i \in [n]$  is 1, we have the following trivial bound for all  $k \in \overline{\text{Br}_i^\delta(\mathbf{x})}$ :

$$v_i(\mathbf{x}')_k - v_i(\mathbf{x})_k + \max_{l \in \text{Br}_i^\delta(\mathbf{x})} \{(v_i(\mathbf{x}))_l\} \leq 2.$$

Substituting these two bounds into Equation (34) gives, for all  $k \in \overline{\text{Br}_i^\delta(\mathbf{x})}$ :

$$Z(\mathbf{x}, \mathbf{x}', \epsilon, k) \leq -\delta + \epsilon \cdot 2.$$

Thus, for each  $k \in \overline{\text{Br}_i^\delta(\mathbf{x})}$ , we have that  $Z(\mathbf{x}, \mathbf{x}', \epsilon, k) \leq 0$  whenever:

$$-\delta + \epsilon \cdot 2 \leq 0,$$

and this is equivalent to:

$$\epsilon \leq \frac{\delta}{2}.$$

This inequality holds by the definition of  $\epsilon$ , so we have  $Z(\mathbf{x}, \mathbf{x}', \epsilon, k) \leq 0$  for all  $k \in \overline{\text{Br}_i^\delta(\mathbf{x})}$ , which then implies that  $\Lambda_i^\delta(\mathbf{x}, \mathbf{x}', \epsilon) \leq 0$ .  $\square$

Next we consider the term  $u_i(\mathbf{x}' - \mathbf{x})$  in Equation (31). The following lemma provides a simple lower bound for this term.

**Lemma 25** *For all  $i \in [n]$ , we have  $Df_i^\delta(\mathbf{x}, \mathbf{x}') - 1 \leq u_i(\mathbf{x}' - \mathbf{x})$ .*

*Proof* For  $u_i(\mathbf{x}' - \mathbf{x})$  we have the following:

$$\begin{aligned} u_i(\mathbf{x}' - \mathbf{x}) &= u_i(x'_i - x_i, \mathbf{x}' - \mathbf{x}) \\ &= u_i(x'_i, \mathbf{x}' - \mathbf{x}) - u_i(x_i, \mathbf{x}' - \mathbf{x}) \\ &= u_i(x'_i, \mathbf{x}') - u_i(x'_i, \mathbf{x}) - u_i(x_i, \mathbf{x}') + u_i(x_i, \mathbf{x}). \end{aligned} \quad (35)$$

Recall from (8) that

$$Df_i^\delta(\mathbf{x}, \mathbf{x}') = \max_{k \in \text{Br}_i^\delta(\mathbf{x})} \{(v_i(\mathbf{x}'))_k\} - u_i(x_i, \mathbf{x}') - u_i(x'_i, \mathbf{x}) + u_i(x_i, \mathbf{x}).$$

We can see that (35) and (8) differ only in terms  $u_i(x'_i, \mathbf{x}')$  and  $\max_{k \in \text{Br}_i^\delta(\mathbf{x})} \{(v_i(\mathbf{x}'))_k\}$  respectively. We know that  $\max_{k \in \text{Br}_i^\delta(\mathbf{x})} \{(v_i(\mathbf{x}'))_k\} \leq 1$ . Then, we can see that  $Df_i^\delta(\mathbf{x}, \mathbf{x}') - 1 \leq u_i(\mathbf{x}' - \mathbf{x})$ .  $\square$

Recall that  $\mathcal{D} = \max_{i \in [n]} Df_i^\delta(\mathbf{x}, \mathbf{x}')$  and  $f_{new} = f(\bar{\mathbf{x}})$  and  $f = f(\mathbf{x})$ . We can now apply the bounds from Lemma 24 and Lemma 25 to Equation (31) to obtain:

$$\begin{aligned} f_{new} - f &\leq \max_{i \in [n]} \left\{ \epsilon(Df_i^\delta(\mathbf{x}, \mathbf{x}') - f(\mathbf{x})) - \epsilon^2(Df_i^\delta(\mathbf{x}, \mathbf{x}') - 1) \right\} \\ &\leq \max_{i \in [n]} \left\{ \epsilon(Df_i^\delta(\mathbf{x}, \mathbf{x}') - f(\mathbf{x})) - \epsilon^2(\mathcal{D} - 1) \right\} \\ &= \epsilon(\mathcal{D} - f) + \epsilon^2(1 - \mathcal{D}). \end{aligned}$$

This completes the proof of Lemma 11.