Deep Reinforcement Learning and Influenced Games

Colin Brady, Rica Gonen, and Genadi Rabinovich

Abstract

Game designers attempt to construct games that influence players to select specific strategies. However, players are not always rational and can play unpredictably. We provide a mechanism to influence behavior in arbitrary normal-form games, even when players are not rational. Given a game and a desired outcome, our algorithm calculates the nearest payoffs that will play as the game designer desires. Our mechanism is based on a deep reinforcement-learning method that models the players' real-world behavior. The mechanism is general and can find solutions in other domains where problems are expressible as classification problems. Finally, we train the algorithm with a combination of real-world and synthesized game data and test the mechanism's performance against a panel of classifiers.

1 Introduction

Game theory examines how rational agents make strategic decisions. It assumes that agents aim to maximize their own utility and analyzes how to find the best strategies for them, known as equilibria. However, these equilibria may not always be the best outcome for players, such as when cooperation would be beneficial. Additionally, the assumption of rationality may not hold in certain scenarios, such as when trust is a factor. An example of this is the prisoner's dilemma, where two criminals are interrogated separately, and each must decide whether to cooperate with their accomplice or betray them to reduce their own sentence. In this situation, cooperation is not an equilibrium as neither criminal can trust the other not to betray them, despite it being in their best interest.

The assumption of rationality may be appropriate in situations where large economic interests are at stake, such as spectrum auctions involving large companies, where there is an incentive to optimize decision-making. However, the perfect rationality assumption may lead to suboptimal outcomes in scenarios with tight time constraints or that involve less sophisticated participants.

One example of a situation where a designer wants exploit irrational behavior is the freemium business model, which is widely used in the mobile gaming industry. In this model the product has some balance between free content and in-app purchases to ensure that a portion of the player base is willing to spend money, while the rest continue to enjoy the product without making purchases. This balance establishes the equilibrium for the players. In this context, game designers create incentives for players and profit when players don?t follow the equilibrium strategy. Essentially, the designers influence the underlying normal form game (NFG) to produce alternative outcomes.

To deal with players who do not behave rationally, the behavioral game theory literature has developed a range of models to predict human behavior in strategic settings. A number of results predicting actual behavior, such as [9, 28, 7, 4, 27, 26, 2], choose the unrepeated, simultaneous-move setting, which is typically represented as normal-form games (NFG)[1]. NFGs have the advantage that they capture many common interactions while remaining conceptually straightforward.

[1]Normal-form game is a description of a game by way of a matrix. The normal-form representation of a game includes all strategies, and their corresponding payoffs, for each player.

Traditional machine learning techniques, e.g., Scalar Vector Machines (SVM) [5], Multi-Layer Perceptrons (MLP) [17, 13], Random Forest [11], do not typically perform well when predicting player behavior. However, deep learning can dramatically improve predictive accuracy, and yet a regular feed-forward neural network performed poorly when applied to NFG matrices by overfitting training data. However, [9] introduced a novel neural network architecture invariant to the input size and makes good predictions.

In some situations, predicting players' behavior may not suffice, such as competing in elections, making policy changes, and designing contracts (e.g., [8]). One may prefer to go beyond prediction in these cases and develop the capability to influence the players' behavior. Influencing builds on predicting, as influencing requires modeling players' likely actions to steer them toward a specific goal. Though influencing game behavior is a natural next step in predicting players' behavior, research in this direction is rarely taken. Though [21] recently investigated influencing players' behavior. [21] proposed a mechanism for incentivizing behavior in extensive-form games. It showed that it is P-complete to determine whether an optimal deposit scheme exists for a finite game of perfect information. The result also showed how to use payments to change such a game's equilibrium.

In the area of mechanism design, we focus on studies pertinent to the prac- tical adaptation of games and the characterization and optimization of human behavior. These works provide a meaningful backdrop against which to under- stand and position our own research. Details can be found in appendix.

1.1 Our Contribution

In this paper, we focus our attention on normal-form games and go beyond prediction. We present a generic mechanism called GUIDE (**G**ame Infl**U**enc**I**ng through **D**eep L**E**arning) to incentivize behavior in arbitrary NFGs by finding the nearest game that will play as desired by the game designer. GUIDE supports both pure and mixed-strategy NFGs. Given a game and a desired outcome, our algorithm calculates the nearest payoffs that will cause players to play the game as intended by the designer. The mechanism is based on [15]'s Deep Q-Learning (DQN) algorithm and models players' actual behavior.

GUIDE vectorizes a set of game data in the form of NFGs and builds mathematically well-defined boundaries for areas containing NFGs with the same outcome. The algorithm then finds the game nearest to the game provided by the designer that will play as requested.

Our result adapts the approach to finding classification boundaries in [16] and proposes solutions to problems encountered therein. [16] addressed the issue of dividing a space using genetic algorithms with hyperplanes. We adapted [16]'s approach by using a DQN, and we deal with [16]'s stopping problem by incorporating a recursive separation technique that ensures the system always converges on a solution. We also provide improved stopping criteria that ensure better classification boundaries.

We adopt [9]'s NFG matrix representation for encoding NFGs as data. Each data point consists of an NFG together with the players' joint actions. We encode the played actions as a classification attached to each NFG. Thus, we have a basis for building boundaries for areas that isolate NFGs with the same classification.

Though [21] recently investigated influencing players' behavior, they focus their study on extensive-form games while we focus our attention on NFGs. Unlike [21], our mechanism will work even when players do not play equilibrium strategies.

We also evaluated our system using a combined dataset of actual game data from [3, 4, 7, 9, 10, 14, 25, 23, 24]. Since, to the best of our knowledge, there are no similar mechanisms to benchmark against, we tested the agreement of our results against a panel of ML classifiers such as SVM [5], MLP [17, 13], and Random Forest [11].

Our technique is demonstrated using NFGs, but it is applicable to other domains where problems can be formulated as classification problems.

1.2 Organization

Section 2 covers the notation and background concepts from game theory and deep reinforcement learning needed for our algorithm. Section 3 explains the mechanism and presents key algorithms. Section 4 shows experimental results and performance evaluation. We conclude and discuss future directions in Section 5. More detail is provided in the appendix in the supplemental material.

2 Preliminaries and Notation

This section provides the necessary background and notation. We discuss normal-form games, hyperplanes, and polytopes and provide relevant notation.

2.1 Normal-Form Games

Let L be a set of players in a normal form game (NFG) where $|L| = l$. Let A^t be the set of actions available for player $t \in L$[2]. Let $a_i^t \in A^t$ be the i_{th} action of player $t \in L$.

An NFG can be represented by a matrix. Each matrix axis represents the actions available to a given player. Each cell expresses the payout from a combination of selected actions. For example, a two-player NFG where each player has two possible actions is represented as follows:

$$\begin{array}{cc} & \begin{array}{cc} a_1^2 & a_2^2 \end{array} \\ \begin{array}{c} a_1^1 \\ a_2^1 \end{array} & \begin{bmatrix} r_{1,1}^1, r_{1,1}^2 & r_{1,2}^1, r_{1,2}^2 \\ r_{2,1}^1, r_{2,1}^2 & r_{2,2}^1, r_{2,2}^2 \end{bmatrix} \end{array}$$

Where $r^t_{a^1_{i_1},..,a^l_{i_l}}$ is the t_{th} player's reward for a specific joint action $(a^1_{i_1},..,a^l_{i_l})$.

NFG Vector A NFG matrix can be placed in a coordinate space by converting it to a vector. A NFG, M, of size $\prod_{t=1}^{l} |A_t|$ can be transformed into a vector of length $l \cdot \prod_{t=1}^{l} |A^t|$. For example, for a two-player NFG where each player has two possible actions, the vector is $(r_{1,1}^1, r_{1,2}^1, r_{2,1}^1 r_{2,2}^1, r_{1,1}^2, r_{1,2}^2, r_{2,1}^2, r_{2,2}^2)$.

Definition 2.1 (NFG Distance)**.** The distance between two NFGs (of the same dimension) V and $\bar{V}$ is defined by the Euclidean distance:

$$\|V, \bar{V}\| =$$

$$\sqrt{\Sigma_{t=1}^{l} \ \Sigma_{a^1_{i_1} \in A^1} ... \Sigma_{a^l_{i_l} \in A^l} (r^t_{a^1_{i_1},..,a^l_{i_l}} - \bar{r}^t_{a^1_{i_1},..,a^l_{i_l}})^2}$$

[2] A^t may also be a set of mixed strategies

2.2 Classification

NFG Vectors are classified by the actions participants choose when playing the given NFGs. Let $C = (a^1_{i_1}, .., a^l_{i_l})$ denote an NFG's classification and let $\hat{C}$ denote the game maker's desired classification.

In the case of 2-player games for an $|A^1| \times |A^2|$ matrix, for row player actions, we define $|A^1|$ classes, while for column player actions, we define $|A^2|$ classes. In the case where we are interested in studying joint actions, we will define $|A^1| \times |A^2|$ classes - one per NFG cell.

From the example matrix provided earlier, in the case of a row player, we have two possible classes: a^1_1 and a^1_2. In the case of a joint action of both row and column players, we have four classes: $a^1_1a^2_1, a^1_1a^2_2, a^1_2a^2_1$ and $a^1_2a^2_2$, and each cell of the matrix represents a separate combination of actions.

For an l-player matrix, the space dimension is $R^{l\times(|A^1|\times\ldots\times|A^l|)}$ or just $R^{l\times(A^l)^l}$ if all l players have the same A^l number of possible actions. In the matrix above, for 2-players with two actions each, we have $2 * 2^2 = 8 \rightarrow R^8$).

2.3 Hyperplanes

Our system uses hyperplanes to define classification boundaries. A normal vector from the origin can completely determine a hyperplane. For example, in R^2, a hyperplane is a simple line which is defined as $x_1 * cos(\alpha) + x_2 * cos(\beta) = d$. The definition of a hyperplane is illustrated in appendix 1.1.

Linking this to the dimension of an NFG matrix transformed to a vector of dimension, R^n, as described in 2.1, we get the following defining equation.

Definition 2.2 (Hyperplane)**.** Given n, the dimension of the enclosing space, d, the length of the normal vector to the hyperplane from the space's origin, and $\alpha_1, \alpha_2, ..., \alpha_{n-1}, \alpha_n$, angles of the normal vector to the corresponding axis of the space. A hyperplane can be expressed as:

$$\Sigma^n_{i=1} x_i * cos(\alpha_i) = d \tag{1}$$

Hyperplane Rotation The hyperplane definition allows us to specify a hyperplane as a tuple $(\alpha_1, \alpha_2, ..., \alpha_{n-1}, \alpha_n, d)$. Hence, a hyperplane can be rotated in space by changing one of its angles $\alpha_1, ..., \alpha_n$, taking into consideration the hyperplane equation coefficients, it is known that [22]:

$$\Sigma^n_{i=1} cos^2(\alpha_i) = 1 \tag{2}$$

Therefore we can express the hyperplane equation with coefficients b_i as follows: $\Sigma^n_{i=1} b_i \cdot x_i = d$.

Given a set W of NFGs transformed to vectors of dimension m, $|W| = w$. We want to insert k hyperplanes to separate regions of NFGs with the same classification. Each of the k hyperplanes is constructed with b_{ij} coefficients of dimension m where j indexes the NFG vector dimension and i indexes the k hyperplanes.

We denote by M a matrix of w NFGs of dimension m. $M \in \mathcal{R}^{w\times m}$. For example, given $w = 5$ NFGs transformed to vectors of dimension $m = 8$, i.e., five NFGs of 2-players with two actions each, we can describe matrix M in the following way:

$$\begin{bmatrix} x_1^1 & x_2^1 & x_3^1 & x_4^1 & x_5^1 & x_6^1 & x_7^1 & x_8^1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1^5 & x_2^5 & x_3^5 & x_4^5 & x_5^5 & x_6^5 & x_7^5 & x_8^5 \end{bmatrix}$$

where $x_1^1 = r_{1,1}^1, x_2^1 = r_{1,2}^1, x_3^1 = r_{2,1}^1, x_4^1 = r_{2,2}^1, x_5^1 = r_{1,1}^2, x_6^1 = r_{1,2}^2, x_7^1 = r_{2,1}^2, x_8^1 = r_{2,2}^2$ etc.

We denote by H a matrix of the NFGs coefficients of dimension m for the k hyperplanes. $H \in \mathcal{R}^{m \times k}$. For example, given $k = 5$ hyperplanes and NFGs coefficients of dimension $m = 8$, we can describe matrix H in the following way:

$$\begin{bmatrix} b_{1,1} & b_{2,1} & b_{3,1} & b_{4,1} & b_{5,1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ b_{1,8} & b_{2,8} & b_{3,8} & b_{4,8} & b_{5,8} \end{bmatrix}$$

We denote by D a matrix of the d_is, the length of the normal vectors to the k hyperplanes from the space's origin for each of the w NFGs transformed to vectors. $D \in \mathcal{R}^{w \times k}$. For example, given $k = 5$ hyperplanes and $w = 5$ NFGs transformed to vectors, we can describe the 5×5 matrix D in the following way:

$$\begin{bmatrix} d_1 & d_2 & d_3 & d_4 & d_5 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ d_1 & d_2 & d_3 & d_4 & d_5 \end{bmatrix}$$

2.4 Polytopes

A polytope is a geometric object with flat sides that generalizes three-dimensional polyhedrons to any number of dimensions. Polytopes are identified by the set of hyperplanes that define their bounds. Polytopes can be open on one or more sides.

Each k hyperplane, defined as $h_i(x) = d_i$, splits the NFGs' space into two regions as follows:

$$\bar{k_i} : h_i(x) - d_i < 0$$
$$k_i : h_i(x) - d_i > 0$$

Each polytope is identified by the set of hyperplane sides ($\bar{k_i}$ *or* k_i) that define its bounds.

Polytope Membership Our algorithm needs to find regions of games with a desired classification. This requires determining which NFGs reside within a given polytope.

The following matrix equation provides us with the position of each classified NFG relative to each hyperplane:

$$P = M \times H - D$$

It follows that $P \in \mathcal{R}^{w \times k}$ where the sign of p_{ij} (positive or negative) specifies the position of NFG j relative to hyperplane i. This means that each row i in P represents the position of NFG j relative to all hyperplanes. This allows us to group NFGs belonging to the same polytopes. We calculate the array of signs +/- relative to defined hyperplanes where sign $-$ points to $\bar{k_i}$ and $+$ points to k_i, and thus the set of these signs points to the set of hyperplane side specifications which identifies a polytope.

We define the set of polytopes as Y.

3 The Mechanism

Our method operates in two phases, building classification boundaries and remapping an NFG, denoted by G, to play as requested by the game maker.

The first phase builds classification boundaries using an input set, W, of NFGs with the players' actions attached. The input NFGs are vectorized, and classifications are created from the players' actions. If an NFG was played multiple times, the most common outcome is selected to classify the NFG. In case of a tie we randomly select a winner. A reinforcement learning algorithm then constructs k hyperplanes, defined by H and D, to isolate regions where NFGs in W share the same classification. Additional steps are taken to ensure homogeneously-classified polytopes, as discussed later in this section.

The second phase remaps G so that it will play with classification $\hat{C}$, as requested by the game maker. This is done by finding the polytope nearest to G with classification $\hat{C}$ using the distance formula given in definition 2.1. G is then mapped to the closest point on the target polytope, shifted a small distance toward the target polytope's interior, and provided to the game maker as $\hat{G}$.

3.1 Build Bounding Polytopes

This section explains how to split the space into regions with the same classification using a reinforcement-learning algorithm and a separation algorithm. The initial space is treated as an unbounded polytope and the process is repeated recursively on each mixed-classification polytope until no new homogenous polytopes can be generated. Any remaining mixed-classification polytopes are then resolved using the separation algorithm.

Polytope building was developed using keras-rl [14]'s DQN, a state-of-the-art deep reinforcement learning algorithm. Specifically, we used [15]'s agent DQN, which combines Q-learning with a deep convolutional neural network. The DQN is structured as follows:

- Input relu layer of size $k * (m + 1)$. The input layer captures the current position of the hyperplanes as there are k hyperplanes with orientation defined in m dimensions and one additional node per hyperplane to capture the distance of each hyperplane from the origin.
- Three hidden relu layers, 64 units each.
- Output softmax layer of size $k * (m + 1) * 2$. The output layer independently expresses the increment or decrement to apply to each hyperplane to change its location in space.

The RL algorithm is presented as a Markov Decision Process, where the state space is represented by H and D. The algorithm can choose to rotate a hyperplane or move it to/from the origin, with the probability of each action determined by past success and subsequent actions. The reward function evaluates the success of each action and favors regions with homogeneous classifications.

The RL algorithm selects a hyperplane and action according to a probability distribution, ρ, which is initialized to a uniform distribution and changes as information is learned. The RL algorithm rotates hyperplanes by selecting one of the coefficients from equation 1 according to ρ. Then, the algorithm changes the selected coefficient by some fraction and adjusts the other coefficients to observe equation 2. Hyperplanes are moved to/from the origin by adding/subtracting a movement increment to/from a hyperplane's distance from the origin, d.

The RL algorithm calculates its reward by computing an isolation score S_t for each polytope $Y_t \in Y$, $1 \leq t \leq |Y|$ where y_t is the number of NFGs bounded by Y_t. The isolation

score is computed as follows:

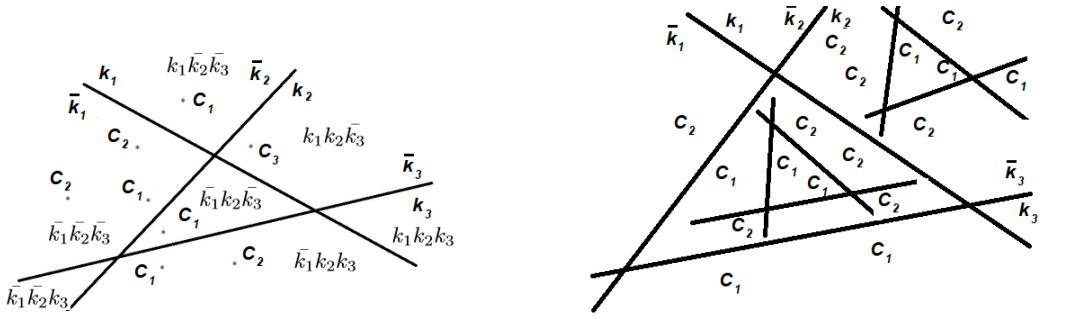
$$S_t = \begin{cases} y_t : \text{if all NFGs in } Y_t \text{ have the same classification} \\ 0 : \text{otherwise} \end{cases}$$

and $S_{Total} = \sum S_t$ is the total reward of the state space.

Although the number of polytopes can reach 2^k for k hyperplanes, practically, this is not the case. Consider that we are only interested in non-empty polytopes whose number is limited by the number of input NFGs. Thus, the RL algorithm has no incentive to create more than w polytopes.

Figure Figure 1(a) demonstrates aspects of reward calculation. The polytope referenced as $\bar{k_1}\bar{k_2}\bar{k_3}$ isolates two instances of C_2 and one instance of C_1, which means that we have misclassification here, and the reward for this polytope will be 0. The same is true for the polytope referenced by $\bar{k_1}k_2k_3$, where we have one of C_1 and C_2, meaning that here too, we have a misclassification, and thus, the reward is 0. Polytopes $\bar{k_1}k_2\bar{k_3}$ and $k_1\bar{k_2}\bar{k_3}$ isolate a single instance of C_1 each – a perfect classification with a reward of 1 each. The same is true for polytope $k_1k_2\bar{k_3}$ with a single instance of C_3; it is a perfect classification with a reward of 1. Polytopes $k_1k_2k_3$ and $\bar{k_1}\bar{k_2}k_3$ have no instances. The total reward for the state (current hyperplane orientation) is:
$S_{Total} = S_{\bar{k_1}\bar{k_2}\bar{k_3}} + S_{\bar{k_1}k_2k_3} + S_{\bar{k_1}k_2\bar{k_3}} + S_{k_1\bar{k_2}\bar{k_3}} + S_{k_1k_2\bar{k_3}} = 0 + 0 + 1 + 1 + 1 = 3.$

After the initial RL algorithm is called, we continue recursively to apply the reinforcement-learning (RL) algorithm to each mixed-classification polytope as is illustrated in Figure 1(b). For an example of a mixed-classification polytope see appendix 1.2. The RL polytope bounding algorithm is contained in appendix 1.4.



(a) Polytope Reward Calculation

(b) RL Recursive Call on Mixed-Classified Polytopes

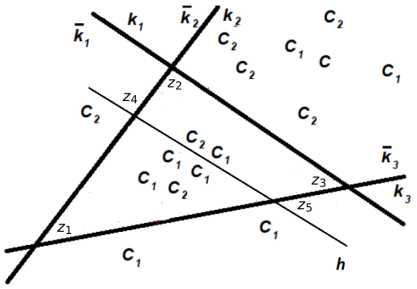
Figure 1: Polytope Examples

As an optimizer can converge on a local minimum, the RL algorithm may reach a state where it cannot create new homogeneously-classified polytopes. See Figure 5 for illustration. This can happen if the RL algorithm cannot find a proper hyperplane orientation or if the selected number of hyperplanes is insufficient for the given NFGs. We apply the separation algorithm to handle this case.

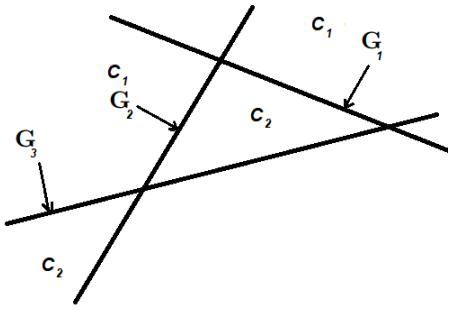
3.1.1 The Separation Algorithm

The separation algorithm recursively subdivides a mixed-classification polytope into multiple homogeneous polytopes. The algorithm subdivides a region by creating a hyperplane

between two points in the region. The separation algorithm is then applied to any mixed-classification polytope resulting from the division, and so on, until homogeneously-classified regions remain. For example, applying the separation algorithm to the mixed classification polytopes $\bar{k_1}k_2\bar{k_3}$ and $k_1k_2\bar{k_3}$ in Figure 5 results in Figure 2(a). In the worst case, the algorithm will require $\bar{w}$ iterations to converge on a solution where $\bar{w}$ is the number of NFGs in a given heterogeneously classified polytope. The separation algorithm can be found in appendix 1.3.



(a) The Separation Algorithm generates a hyperplane h to create polytopes $z_1z_4z_5$ and $z_2z_3z_5z_4$.



(b) Finding points nearest to a polytope with class C_2

Figure 2: Separation Algorithm Illustrations

3.2 NFG Remapping

Given an input NFG matrix, G, we want to find the minimal modification that will cause players to play the actions defined by classification $\hat{C}$. We remap G to play $\hat{C}$ by locating the nearest polytope that isolates NFGs classified as $\hat{C}$ and then find the closest point on that polytope (Figure Figure 2(b)), G'. To increase the probability that the modified NFG will play $\hat{C}$, G' is moved toward the point with the steepest probabilty gradient in the incident polytope to create $\hat{G}$.

The algorithm finds the nearest point on a polytope that isolates NFGs classified as $\hat{C}$ by solving a linear programming (LP) problem for each such polytope. The LP solution gives the nearest point to G for each polytope [3], and thus the distance between each polytope and G.

The LP's constraints are defined by the set of constraints concluded from the hyperplane equations that constitute a given polytope. The objective function minimizes the euclidean distance from G to the polytope. For G with coordinates $(X_1, ..., X_m)$ and a set of k hyperplanes:

$$\Sigma_{j=1}^{m} b_{1j} * x_j = d_1$$
$$\vdots$$
$$\Sigma_{j=1}^{m} b_{kj} * x_j = d_k$$

The set of polytopes Y is defined by sets of constraints derived from the k hyperplane equations in the form of $\Sigma_{j=1}^{m} b_{ij} * x_j < d_i$ or $\Sigma_{j=1}^{m} b_{ij} * x_j > d_i$, depending on which side of

[3] In the case the G is already in a relevant polytope, then the algorithm returns G.

a specific hyperplane a polytope is located. The objective function is:

$$min\sqrt{\Sigma_{j=1}^{m}(X_j - x_j)^2}$$

Thus G' is the point with the minimal distance from G among all the solutions found by the set of LPs. Let $\hat{Y}_t$ be the polytope that G' belongs to.

For example, Figure Figure 3(a) illustrates two classes C_1 and C_2 and polytopes bounded by the three hyperplanes:

$$b_{11} * x_1 + b_{12} * x_2 = d_1$$
$$b_{21} * x_1 + b_{22} * x_2 = d_2$$
$$b_{31} * x_1 + b_{32} * x_2 = d_3$$

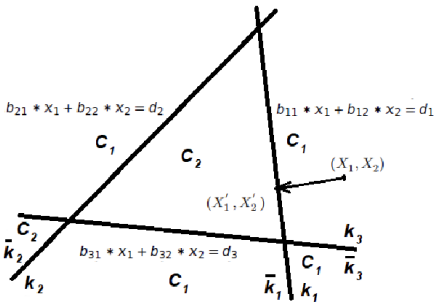
If G is at point (X_1, X_2), and we want to find the nearest point on a polytope classified as C_2, we solve an LP problem for each polytope containing C_2. In this case we solve the LP for $\bar{k_1}\bar{k_2}\bar{k_3}$ and $\bar{k_1}k_2k_3$, under the objective function $min\sqrt{(X_1 - x_1)^2 + (X_2 - x_2)^2}$. For polytope $\bar{k_1}\bar{k_2}\bar{k_3}$ the constraint set derived from the hyperplanes is:

$$b_{11} * x_1 + b_{12} * x_2 < d_1$$
$$b_{21} * x_1 + b_{22} * x_2 < d_2$$
$$b_{31} * x_1 + b_{32} * x_2 < d_3$$

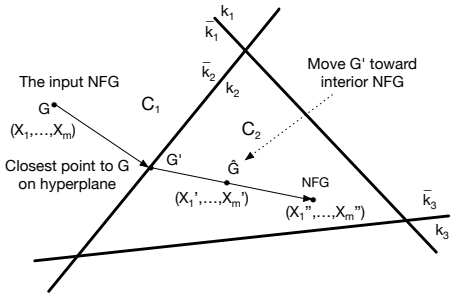
and for $\bar{k_1}k_2k_3$, the constraint set derived from the hyperplanes is:

$$b_{11} * x_1 + b_{12} * x_2 < d_1$$
$$b_{21} * x_1 + b_{22} * x_2 > d_2$$
$$b_{31} * x_1 + b_{32} * x_2 > d_3$$

In this example, the nearest point is from $\bar{k_1}k_2k_3$, and the system solves and returns $(X_1^{'}, X_2^{'})$.



(a) Finding the nearest point on a polytope surface



(b) Improving Remapping Confidence

Figure 3: NFG Remapping Illustrations

G' falls on a polytope surface that also belongs to an adjacent polytope with a different classification. To make $\hat{G}$ play as intended with higher expectation, we move the solution NFG further into $\hat{Y}_t$'s interior. Given point G', we find the NFG in $\hat{Y}_t$ with the steepest probability gradient from G', the point $(X_1^{''}, ..., X_m^{''})$, and move G' toward $(X_1^{''}, ..., X_m^{''})$ and return its' new location as $\hat{G}$. The gradient is calculated as the probability that each

interior point will play $\hat{C}$ divided by the distance between G' and each interior NFG. Figure Figure 3(b) illustrates this operation.

The experimental results include an investigation of the relationship between the remapping confidence and the distance the solution point was moved into the bounding polytope's interior.

3.3 The Algorithm

We now present the main algorithm:

Algorithm 1 GUIDE

procedure MAIN(W,$C_1, ...C_w$, k, G, $\hat{C}$)
Input:
W, the dataset of input NFG matrices
$C_1, ...C_w$ a set of clasifications for each NFG $\in W$
k, number of hyperplanes to divide the vectorized NFGs
G, the NFG to remap
$\hat{C}$, the desired classification for G
Output:
$\hat{G}$ the remapped NGF such that $\hat{G}$ is in polytope $\hat{Y}_t$
with clasification $\hat{C}$.

Build M, i.e. vectorize NFGs in W.
Denote v^* by $\max_{V \in W} \|V, V^o\|$ where V^o is the origion.
Initialize H and D with k hyperplanes by placing them
parallel to each other with $d_i = v^*/i$.
Compute P from H, D, M.
Call RL Bound Regions (see appendix 1.4) with
P to build homogenously-classified polytopes.
if mixed-classification polytopes remain **then**
Apply the Separation Algorithm
Remap G to G' on the (nearest) polytope $\hat{Y}_t$ classified $\hat{C}$.
Improve confidence by moving G' toward the NFG
wih the steepest probability gradient in $\hat{Y}_t$. Call the new location $\hat{G}$.
return $\hat{G}$

4 Experimental Results

This section presents an experiment to evaluate the NFG-remapping system's performance. Since, to the best of our knowledge, there are no comparable mechanisms to benchmark against, we test the agreement of our results with the output of a panel. Our panel consists of known classifiers trained on our dataset of played NFGs. This means the evaluation of our results is only as accurate as the classifiers on the panel. The details of the classier panel are given in the supplemental materials in appendix 1.6. Finally, we assess the effect of moving G' into the interior of the target polytope, $\hat{Y}_t$, to create $\hat{G}$, on the quality of the remapping.

4.1 Experiment Design

We trained GUIDE using datasets from the combined observations of several experimental game theory studies in which participants were paid to select actions in NFG. Participants

did not always play the equilibrium strategy or might have played games with multiple equilibria, so participants' sometimes irrational behavior is captured in the data.

[3, 4, 7, 9, 10, 14, 25, 23, 24]. These studies used the 61, 3×3, 2-player NFGs from the cited datasets. These games were played 4,812 times overall. The results analyze the row player's chosen action, which was named action A, B, and C, respectively. In the dataset, any single NFG was played multiple times with different results. So every NFG had a distribution of classifications rather than a single classification. As noted, this distribution could result from players choosing non-equilibrium actions or playing multiple equilibria. We resolved this by mapping each NFG to the most frequently played action and retaining the probability of that played action. So even if human players played equilibrium actions in a multi-equilibria game, we associate such an NFG with the most frequently played equilibrium action and the probability that the action was played. Thus we note when other equilibria exist but are not frequently played. Lastly, tie-breaking was unnecessary as this dataset did not contain ties between classifications.

We supplemented the actual game data with synthesized data to fill in the gaps between the regions explored in the experimental data. We only synthesized the areas that play actions with high expectation, i.e., the locations that an analysis of each NFG would easily predict. The assumption is that the game maker generates high-probability data from analysis to describe the contour of the space and populates boundary regions with actual experimental data. The synthesized data was created by generating 50,000 uniformly-distributed NFGs and filtering out those that played their preferred action with probability less than 0.9, as classified by NFGnn. The remaining 1000 data points defined well-separated regions, which we confirmed with a t-SNE plot.

The system was evaluated with a set of 101 test NFGs was randomly generated to be used as Gs and remapped in the experiment. The 101 test data points included 61 points taken from the experimental data and another 50 generated points. The 50 synthetic points were randomly created with a uniform distribution. The payout range was selected to match the range of payouts in the datasets from [3, 4, 7, 9, 10, 14, 25, 23, 24].

The following approach was used to estimate the accuracy of our remapping algorithm:

1. We selected several ML classifiers: SVM[5], Random Forest[18, 13], Multi-layer Perceptron[11], and NFGnn[9].
2. The classifiers were trained on the original 4,812 game matrix instances from [3, 4, 7, 9, 10, 14, 25, 23, 24].
3. The accuracy of each classifier was evaluated.
4. Each classifier labeled each of our solution NFGs, $\hat{G}$.
5. The agreement between each classifier and $\hat{C}$ was recorded.
6. The solution NFGs were moved in steps toward the interior of their respective $\hat{Y}_t$s and reevaluated against the benchmark classifiers at each step.

A detailed discussion of how to select the correct value for k when building the polytopes is contained in appendix 1.5.

4.2 Experiment Results

The experimental results showed that the system was surprisingly reliable relative to the performance of the testing classifiers. As the NFGnn classifier demonstrated nearly perfect accuracy, the most interesting comparison is between our mechanisms and NFGnn. As can be seen from the agreement plot with NFGnn, the agreement level of 49% when $\hat{G}$ is close to

G' matches our expectation that solutions on polytope boundaries would be classified as $\hat{C}$ roughly half the time. This is because each boundary also belongs to an adjacent polytope with a different classification. The steady increase in agreement as $\hat{G}$ was moved toward an inner NFG is also expected as $\hat{G}$ is converging on another point already classified as $\hat{C}$. As we wanted to see, the agreement rises sharply toward 100% as we follow the steepest probability gradient in the target polytope. The rapid rise in agreement with the SVM likely reflects that both mechanisms attempt to impose classifications through explicit geometric boundaries.

The detailed experimental results are presented in tables in the supplementary material. The tables are summarized in Figure 4 below. The results show the classifier agreement for each classification vs. the fraction of distance the solution NFG moved between G' and the closest NFG in $\hat{Y}_t$. E.g., 0 indicates that $\hat{G}$ stayed at G' while 1 means that the solution NFG is a previously played NFG. Figure 4 summarizes this relationship and shows that the system can be configured for a preferred accuracy. One can see how the classifier agreement changes as we move $\hat{G}$ further into $\hat{Y}_t$'s interior. As expected, the confidence rises as $\hat{G}$ moves from polytope boundaries and closer to the existing NFGs.

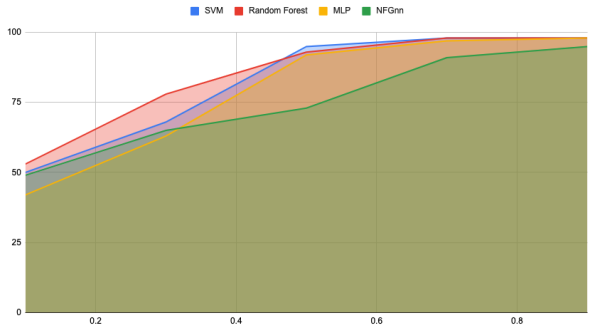


Figure 4: Classifier Agreement on $\hat{C}$ vs % Distance to Nearest Point

We present the case where the system was only trained with real-world data in appendix 1.8. Last but note least, we plan to make our dataset and algorithm code public however the IJCAI double blind policy restricts us from doing so until after the paper is camera ready as publishing this information would breach anonymity.

5 Conclusions and Discussion

We presented GUIDE, a reinforcement-learning-based mechanism for helping game designers create games with desirable outcomes. The influenced games were played by human players, so players did not always play the equilibrium strategy. The mechanism is general enough to apply to other game theory problems. The experimental results showed that the system operated as expected and was reliable relative to the performance of the testing classifiers.

GUIDE was designed and developed to apply to a wide range of problems that can be represented as classification problems. As part of the proof of concept, we executed tests on the Iris [6], and Pen-Digit [1] datasets to see how they would perform with data other than NFGs. Our mechanism provided similar or better performance on these data sets. A summary of the results can be found in the supplementary materials.

Simplifications were made to increase the clarity of this result and are left open for future exploration. This includes allowing the RL algorithm to choose the optimal number

of hyperplanes and making better use of probabilities in the base model to improve the solution's accuracy. We leave these questions for future exploration.

References

[1] Fevzi Alimoglu and Ethem Alpaydin. Methods of combining multiple classifiers based on different representations for pen-based handwritten digit recognition. In *Proceedings of the Fifth Turkish Artificial Intelligence and Artificial Neural Networks Symposium (TAINN 96*. Citeseer, 1996.

[2] Branislav Bosansky, Viliam Lisy, Marc Lanctot, Jiri Cermak, and Mark Winands. Algorithms for computing strategies in two-player simultaneous move games. *Artificial Intelligence*, 237, 04 2016. doi: 10.1016/j.artint.2016.03.005.

[3] Chong JK. Camerer CF, Ho TH. A cognitive hierarchy model of games. *The Quarterly Journal of Economics*, 119(3):861–98, Aug 2004.

[4] D.J. Cooper and J.B. Van Huyck. A cognitive hierarchy model of games. *Journal of Economic Theory*, 110(2):290–308, 2003.

[5] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. doi: 10.1007/BF00994018. URL https://doi.org/10.1007/BF00994018.

[6] R.A. Fisher. Use of multiple measurements in taxonomic problems. *Annual Eugenics*, 7.

[7] D. Fudenberg and A. Liang. Predicting and understanding initial play. *American Economic Review*, 109(12):4112–41, 2019.

[8] Marilyn George and Seny Kamara. Adversarial level agreements for two-party protocols. *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, 2020.

[9] Wright J.R. Hartford, J.S. and K. Leyton-Brown. Deep learning for predicting human strategic behavior. In *In Advances in Neural Information Processing Systems*, pages 2424–2432, 2016.

[10] Stahl D.O. Haruvy, E. and P.W. Wilson. Modeling and testing for heterogeneity in observed strategic behavior. *Review of Economics and Statistics*, 83(1):146–157, 2001.

[11] Tin Kam Ho. Random decision forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, ICDAR '95, pages 278–282, M, 1995. IEEE Computer Society. ISBN 0-8186-7128-9.

[12] Ledyard J. O. Kalai, E. Repeated implementation. *Journal of Economic Theory*, 83 (2):308–317, 1998.

[13] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. *Efficient BackProp*, pages 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-35289-8. doi: 10.1007/978-3-642-35289-8_3. URL https://doi.org/10.1007/978-3-642-35289-8_3.

[14] Plappert M. Keras-r. https://github.com/keras-rl/keras-rl.

[15] Kavukcuoglu K. Silver D. Graves A. Antonoglou I. Wierstra D. Mnih, V. and M. Riedmiller. Playing atari with deep reinforcement learning. In *arXiv:1312.5602*, 2013.

[16] Bandyopadhyay S. Pal, S.K. and C.A. Murthy. Genetic algorithms for generation of class boundaries. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 28(6):816–328, 1998.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press, Cambridge, MA, USA, 1986. ISBN 026268053X.

[18] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and James L. Mcclelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[19] Wang Y. Saberi, A. The complexity of optimizing over a simplex, hypercube or sphere: a short survey. *Discrete Optimization*, 16:33–46, 2015.

[20] Lesser V. Sandholm, T. Leveled-commitment contracting: A backtracking instrument for multi-agent systems. *AI Magazine*, 22(1):89, 2001.

[21] N. Schwartzbach. Payment schemes from limited information with applications in distributed computing. In *In EC'22*, pages 129–149, 2022.

[22] M.R. Spizel. *Vector Analysis*. McGraw-Hill, New York, NY, 1959.

[23] Dale Stahl and Paul Wilson. Experimental evidence on players' models of other players. *Journal of Economic Behavior & Organization*, 25(3):309–327, 1994. URL https://EconPapers.repec.org/RePEc:eee:jeborg:v:25:y:1994:i:3:p:309-327.

[24] Dale Stahl and Paul Wilson. On players' models of other players: Theory and experimental evidence. *Games and Economic Behavior*, 10(1):218–254, 1995. URL https://EconPapers.repec.org/RePEc:eee:gamebe:v:10:y:1995:i:1:p:218-254.

[25] R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction.* MIT Press, Cambridge, Massachusetts, 2018.

[26] James R. Wright and Kevin Leyton-Brown. Beyond equilibrium: Predicting human behavior in normal-form games. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, page 901–907. AAAI Press, 2010.

[27] James R. Wright and Kevin Leyton-Brown. Level-0 meta-models for predicting human behavior in games. In *Proceedings of the Fifteenth ACM Conference on Economics and Computation*, EC '14, page 857–874, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450325653. doi: 10.1145/2600057.2602907. URL https://doi.org/10.1145/2600057.2602907.

[28] J.R. Wright and K. Leyton-Brown. Level-0 models for predicting human behavior in games. *Journal of Artificial Intelligence Research*, 64:357–383, 2019.

6 Appendix

6.1 Mechanism Design Related Work

[20] allows for flexibility and adaptability within the game by introducing leveled-commitment contracting, a concept that provides a way for agents in a multi-agent system to make contracts they can de-commit from, albeit with a cost. However, [20] isn't concerned with player influence but focuses on creating flexible contracts among autonomous agents in a multi-agent system. Both [20] and our approach deal with strategic behavior, though they differ in terms of goals and mechanisms. [12] provides a deeper understanding of the strategies and equilibria that may arise in repeated economic interactions, when the same economic game is played repeatedly, and suggests that such interactions may lead to more efficient or desired outcomes than one-time games. While [12] uses traditional economic and game theory principles to describe outcomes theory, our result focuses on actual outcomes expressed in data. [19] delves into the complexities associated with optimizing over specific geometric shapes common to game theory optimization, namely the simplex, hypercube, and sphere, and suggests strategies and approaches for solving them. In contrast, our result focuses on a practical solution to optimize player behavior in games embedded in a hypercube.

7 Separation Iteration Illustration

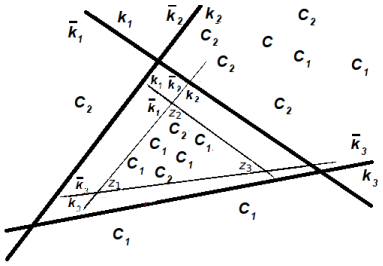


Figure 5: At some RL separation iteration, hyperplanes could fail to split instances, and the process will not advance.

Colin Brady
The Open University of Israel
Email: cbrady321@gmail.com

Rica Gonen
The Open University of Israel
Email: ricagonen@gmail.com

Genadi Rabinovich
The Open University of Israel
Email: genadir@gmail.com