IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 25 July 2024, accepted 9 August 2024, date of publication 15 August 2024, date of current version 27 August 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3443873

RESEARCH ARTICLE

Deep Reinforcement Learning and Influenced Games

C. BRADY, R. GONEN, AND G. RABINOVICH
Department of Mathematics and Computer Science, OUI, Ra'anana 43107, Israel
Corresponding author: R. Gonen (ricagonen@gmail.com)

ABSTRACT In game design, creators strive to guide players toward specific strategies through the game's mechanics. This assumes a level of predictability in player behavior, though in reality, players often deviate from expected rational strategies. We provide an ensemble mechanism to influence behavior in arbitrary normal-form games, even when players are not rational. Our algorithm adjusts the game's reward structure to encourage gameplay that aligns with the designer's intentions. At the algorithm's core is a deep reinforcement learning technique that learns to model players' real-world behavior. This mechanism is versatile and applicable beyond game design; it can be employed in various fields where one can frame problems as classification tasks. Incorporating actual gameplay data into the training process allows our algorithm to acquire a practical understanding of player decisions. The efficacy of our mechanism is evaluated by testing the mechanism's performance against a panel of classifiers, which includes a support vector machine, random forest, and multi-layer perceptron.

INDEX TERMS Influencing games, normal-form games, reinforcement learning.

I. INTRODUCTION

Game theory traditionally assumes agents are rational, striving to maximize their utilities and seeking equilibria. In reality, cooperation can be beneficial, and full rationality does not always prevail. While the rationality assumption works for scenarios involving significant economic stakes, like spectrum auctions, it can falter under time constraints or with less sophisticated players. In these situations, seemingly irrational behavior can be exploited for gain. The freemium business model, prevalent in mobile gaming, exemplifies this: players are incentivized toward a balance of free content and in-app purchases, with profit stemming from deviations from the equilibrium strategy.

Certain aspects of automated mechanism design [1] grapple with irrational players by devising mechanisms with equilibriums that exhibit resilience to irrational conduct, e.g., Bayesian Robust Mechanisms [2]. In contrast, our study integrates machine learning components to steer player behavior, irrespective of its rational or irrational nature, but does not aim to comprehend the equilibrium or manage games characterized by such equilibriums. Consequently, it is plausible that some games formulated under our model may lack an equilibrium entirely.

Handling non-rational players has spurred the development of behavioral game theory models for predicting strategic human behavior [3], [4], [5], [6], [7], [8], [9]. The models frequently focus on unrepeated, simultaneous-move scenarios, aptly represented as normal-form games (NFG).[1] NFGs capture many interactions while remaining conceptually straightforward.

Traditional machine learning techniques, e.g., Support Vector Machines (SVM) [10], Multi-Layer Perceptrons (MLP) [11], [12], Random Forest [13], do not typically predict strategic player behavior well. However, [3] introduced a novel deep-learning neural-network architecture invariant to the input size that makes good predictions.

Influencing behavior is crucial in contexts like elections, policy changes, and contract design [14]. This entails modeling likely player actions to guide them toward specific goals. Recent work by [15], probes this, presenting a mechanism to

The associate editor coordinating the review of this manuscript and approving it for publication was Jiachen Yang.

[1] A normal-form game is a game description via a matrix, detailing all strategies and corresponding payoffs for each player.

incentivize behavior in extensive-form games, demonstrating the *P*-completeness of determining optimal deposit schemes in perfect-information games, and demonstrating the use of payments to alter game equilibrium.

Various research attempts to deal with irrational players in the fields of opponent-shaping, adaptive mechanism design, differentiable economics, and empirical game theory. Reference [16] explores deep reinforcement learning (RL) methods to model and understand players' behavior in such games. Works such as [17] and [18] involve analyzing player behavior and using it to design adaptive mechanisms that shape opponents' strategies. Strategy shaping can involve dynamically adjusting strategies in response to opponent behaviors. On the other hand, [19] focuses on analyzing and designing mechanisms based on empirical data and real-world settings and provides insights into how empirical methods can inform mechanism design in practical scenarios, even when the data is extremely scarce. Reference [15] recently investigated influencing players' behavior in extensive-form games where players play equilibrium strategies. Note that in contrast to these results, our result works with NFGs and seeks to change the game's overall outcome rather than adapt to opponents and is not concerned with equilibria.

GUIDE uses [20]'s Deep Q-Learning (DQL) algorithm to divide the space of possible games into regions with the same expected outcome. GUIDE vectorizes a set of game data in the form of NFGs and uses an ensemble approach [21], [22] to build mathematically well-defined boundaries for areas containing NFGs with the same outcome. The algorithm analyzes the polytopes to find the game nearest to the designer's that will play as requested. We selected DQN for its ability to handle complexity and scale to high-dimensional spaces.

Although our study primarily integrates conventional machine learning components to steer player behavior in Normal-Form Games (NFGs), it is worth noting the broader context in which these technologies operate. Distributed machine learning [23], for instance, offers scalable solutions that could enhance the processing of large-scale game data across multiple servers or nodes without the need to share sensitive data.

Our result adapts [24]'s approach to finding classification boundaries by dividing a space using genetic algorithms with hyperplanes. GUIDE adapts [24]'s approach by using a DQN and solves its stopping problem by incorporating a recursive separation technique with stopping criteria that ensure the system converges on a solution and results in better classification boundaries.

A. OUR CONTRIBUTION

In this paper, 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. GUIDE receives a game from a designer and finds the nearest game[2] that plays as the designer intends. GUIDE supports both pure and mixed-strategy NFGs and applies to other domains where problems can be formulated as classification problems.

Reference [3]'s NFG matrix representation was adopted for encoding NFGs as data. Each data point consists of an NFG together with the players' joint actions represented as a classification. This creates a basis for bounding areas that isolate NFGs with the same classification.

We evaluated GUIDE using a combined dataset of actual game data obtained from [3], [5], [6], [25], [26], [27], [28], [29], and [30]. 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 [10], MLP [11], [12], and Random Forest [13].

It is important to note that GUIDE is indifferent to theoretical equilibria. GUIDE only considers outcomes to the extent that they appear in actual data. The outcome requested by the game designer can be any combination of player actions.

B. ETHICAL CONSIDERATIONS

Our research introduces an ensemble classification technique with potential applications in modifying games represented by Normal-Form Games (NFGs) to align with designers' expectations. While our technique is inherently neutral, its application can lead to various outcomes depending on the user's intent. The technology has the potential to enhance fairness and balance in games, ensuring a better experience for players. However, it also carries risks of misuse, such as manipulating game outcomes for dishonest gain or creating biases in competitive scenarios.

The ethical implications of machine learning technologies in game modification necessitate ongoing discussion and careful consideration. As highlighted by [31], AI ethics should focus on both supporting the socially positive use of these technologies and preventing their misuse. In line with this perspective, we advocate for the responsible application of our technology.

To mitigate potential ethical risks associated with our approach, we propose several strategies:

1) Ensuring transparency by openly disclosing algorithmic modifications to players, allowing them to make informed decisions about their gameplay.
2) Adopting inclusive design practices that involve a diverse range of stakeholders in the development process, which helps ensure fairness and balance in-game mechanics for all players.
3) Regularly monitoring the effects of game modifications on player experience and maintaining consistent standards of fairness.

These strategies align with the principles of fairness, accountability, and transparency in machine learning, as discussed by [32] in their comprehensive survey of the field. Their work emphasizes the importance of ethical

[2]later define in 1.

considerations in the development and deployment of algorithmic systems.

We underscore the obligation of developers and users to apply this technology ethically and to consider the broader consequences of their actions on the game's stakeholders. By promoting these practices, our aim is to contribute positively to the field of game design and ensure that advancements in gaming technology foster an equitable and enjoyable experience for all participants.

C. ORGANIZATION

Section II covers the notation and background concepts from game theory and deep reinforcement learning needed for our algorithm. Section III explains the mechanism and presents key algorithms. Section IV shows experimental results and performance evaluation. We conclude and discuss future directions in Section V. The appendix contains deferred details and introduces experimental results for GUIDE with non-NFG datasets.

II. PRELIMINARIES AND NOTATION

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

See Table 1 for a summary of all notations.

A. NORMAL-FORM GAMES

Let L represent a set of players in an NFG where $|L| = l$. Let A^t be the set of actions available for player $t \in L$.[3] 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:

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

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

1) NFG VECTOR

An NFG matrix can be placed in a coordinate space by converting it to a vector. An 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 $(r^1_{1,1}, r^1_{1,2}, r^1_{2,1} r^1_{2,2}, r^2_{1,1}, r^2_{1,2}, r^2_{2,1}, r^2_{2,2})$.

B. 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. If 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: $a^1_1 a^2_1$, $a^1_1 a^2_2$, $a^1_2 a^2_1$ and $a^1_2 a^2_2$, 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 $2 * 2^2 = 8 \rightarrow R^8$).

C. NFG DISTANCE

GUIDE incentivizes behavior in NFGs by finding the nearest game that will play as the game designer desires. There are various ways to define the "nearest" game. It is reasonable to assume that a game designer would want a game that is close in cost and class to the original game. Therefore, we define the distance of two NFGs as their Euclidean distance and illustrate this choice as aligned with finding a cheaper potential payoff solution out of the options for the designer.

An illustrative example can be found in appendix -A.

Definition 1 (NFG Distance): The distance between two NFGs (of the same dimension) V and $\bar{V}$ is defined by the Euclidean distance:

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

For ease of comprehension, the example provided in appendix -A uses rational players, though our research focuses on players who may not necessarily make rational moves. Nonetheless, the algorithm introduced in Section III handles both rational and irrational behaviors. Also, note that the space of classified NFGs may contain multiple regions with the same classification with a considerable distance between them. This can occur, for instance, when the payoffs of a specific NFG are multiplied by a positive constant, resulting in a new game with the same incentives. In such cases, we will identify the region closest to the given NFG among the multiple regions with the desired classification.

D. 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 that is defined on the axes x_1, x_2 as $x_1 * cos(\alpha) + x_2 * cos(\beta) = d$.

Linking this to the dimension of an NFG matrix transformed to a vector of dimension, R^n, as illustrated by Fig. 1, we get the following defining equation.

[3] A^t may include mixed strategies.

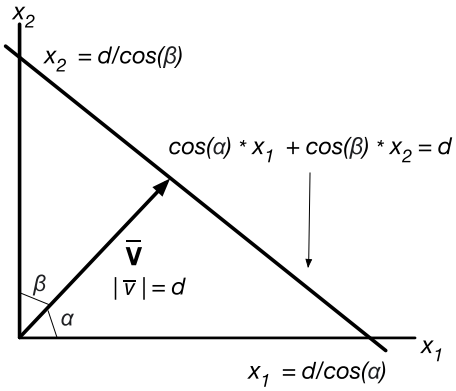


FIGURE 1. The definition of a 2D hyperplane.

Definition 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 $\alpha_1, \alpha_2, \ldots, \alpha_{n-1}, \alpha_n$, angles of the normal vector to the corresponding axis of the space. A hyperplane can be expressed as: $\Sigma_{i=1}^{n} x_i * cos(\alpha_i) = d$.

1) HYPERPLANE ROTATION

The hyperplane definition allows us to specify a hyperplane as a tuple $(\alpha_1, \alpha_2, \ldots, \alpha_{n-1}, \alpha_n, d)$. Hence, a hyperplane can be rotated in space by changing one of its angles $\alpha_1, \ldots, \alpha_n$, taking into consideration the hyperplane equation coefficients, it is known that [33]: $\Sigma i = 1^n cos^2(\alpha i) = 1$.

Therefore we can express the hyperplane equation with coefficients bi as follows:

$$\sum_{i=1}^{n} b_i \cdot x_i = d \tag{2}$$

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. $M \in \mathcal{R}^{w \times m}$. We denote by H a matrix of the NFGs coefficients of dimension m for the k hyperplanes. $H \in \mathcal{R}^{m \times k}$. 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. $D \in \mathcal{R}^{w \times k}$. An example of how to construct M, H, and D is given in appendix -B.

E. 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.

1) 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.

F. DEEP-Q LEARNING

Deep-Q Learning is an advanced reinforcement learning algorithm that integrates classical Q-learning with deep neural networks. This technique allows agents to learn optimal policies for decision-making tasks that involve complex, high-dimensional environments where traditional Q-learning approaches would be impractical.

At its core, DQL utilizes a neural network to approximate the Q-value function, which estimates the expected cumulative rewards for taking each possible action in every given state. This Q-value function is crucial because it helps the agent to evaluate the potential future benefits of its actions, guiding it to make decisions that maximize the cumulative reward over time.

The learning process in DQL involves repeatedly interacting with the environment, collecting experiences in the form of state-action-reward-next state tuples, and using these to update the Q-values predicted by the neural network.

The reward function in DQL provides immediate, quantifiable feedback reflecting the effectiveness of the agent's actions. Mathematically, it is expressed within the Bellman equation used for updating Q-values, (Note that the following notation is specific to this section):

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$$

where:

- r represents the reward received after the agent performs an action a in state s,
- γ is the discount factor, indicating the importance of future rewards,
- $\max_{a'} Q(s', a')$ is the best possible future reward from the next state s',

TABLE 1. Notation summary.

Notation	Meaning
L	A set of players in an NFG.
l	$= \|L\|$.
A^t	The set of actions available for player $t \in L$.
$a_i^t \in A^t$	The ith action of player $t \in L$.
G	An NFG.
C	An NFG classification/game outcome.
$\hat{C}$	The classification/game outcome the game maker desires.
$\hat{G}$	An NFG with expected outcome $\hat{C}$, returned to the game maker as the result of GUIDE.
V	A vectorized NFG.
$\alpha_1, ..., \alpha_n$	The vector of coefficients that define the orientation of a hyperplane.
$\Sigma_{i=1}^{n} x_i * cos(\alpha_i) = d$	Defines a hyperplane in an n-dimensional space with a normal vector to the origin length d.
W	The set of NFGs transformed to vectors of dimension m.
w	$= \|W\|$.
k	The initial number of hyperplanes provided to the separation algorithm.
D	The matrix of normal vector lengths to the k hyperplanes from the space's origin for each of the w NFGs transformed to vectors. $D \in \mathcal{R}^{w \times k}$.
H	The matrix of NFGs coefficients of dimension m for the k hyperplanes. $H \in \mathcal{R}^{m \times k}$.
M	A matrix of w NFGs of dimension m. $M \in \mathcal{R}^{w \times m}$.
Y	The set of polytopes.
P	A matrix that provides the position of each classified NFG relative to each hyperplane. $P = M \times H - D$. $P \in \mathcal{R}^{w \times k}$.
p_{ij} +/-	Specifies the position of NFG j relative to hyperplane i.

- α is the learning rate, controlling the integration of new information.

This formula encapsulates the reward's role: it directly influences the update of Q-values, integrating immediate outcomes with anticipated future benefits, thus steering the agent's learning trajectory toward achieving optimal long-term rewards.

1) NETWORK STRUCTURE

GUIDE's Deep Q-Network (DQN) has the structure:

1) Input Layers - The input layer flattens the input and preprocesses the data for future layers. This layer captures the state of the environment as described by the position of all hyperplanes in space defined by P. Thus, if we have k hyperplanes the state size is $w \cdot k$.
2) Hidden Layers - Three dense, fully connected, hidden layers of 64 units each, each activated by a rectified linear unit (ReLU) function. These layers enable the network to learn intricate patterns and representations of the environment's states and actions.
3) Ouptut Layers - The final dense layer outputs probabilities for each possible action, facilitated by a softmax activation function. This setup allows the agent to select actions probabilistically based on their associated Q-values.

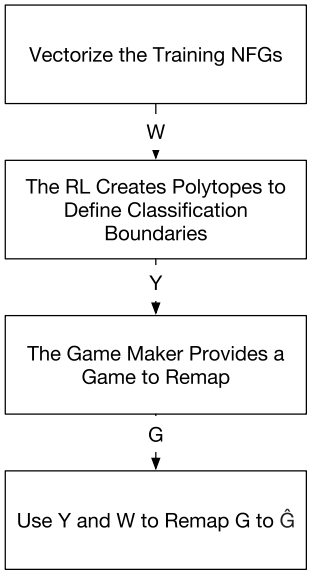


FIGURE 2. Overview of GUIDE stages.

The neural network is configured to use a Boltzmann Q-Policy, which helps in selecting actions in a manner that balances between exploration of new actions and exploitation of known actions. Additionally, the network utilizes experience replay via SequentialMemory to efficiently learn from past experiences by revisiting and learning from them, which enhances the stability and effectiveness of the learning process.

III. THE MECHANISM

The mechanism operates in the stages represented in Fig. 2.

We present the algorithm in two phases, each consisting of two sequential stages from the overview. These phases are 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 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 $\hat{G}$.

A. 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 [27]'s DQN, a state-of-the-art deep reinforcement learning algorithm. Specifically, we used [20]'s agent DQN, which combines Q-learning with a deep convolutional neural network. The DQN is structured as follows:

- Input a fully-connected MLP layer of of size $k * (m + 1)$ with ReLU activation. 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 layers with ReLU activation, 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 H and D represent the state space. 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 the equation in Definition 2 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.

1) THE RL REWARD FUNCTION

This section describes the RL's reward function, which is the core of the ensemble technique for creating high-quality polytopes. We start by formalizing the reward function notation, then presents the reward function, and proceeds to explain the intuition behind the function.

For context, in the function ensemble, the RL positions region-bounding hyperplanes while NFGnn informs the RL on classification probability contours, which aids optimal hyperplane placement. The RL algorithm computes its reward using a quality score S_t for each homogeneous polytope $Y_t \in Y$, $1 \leq t \leq |Y|$, where y_t is the number of NFGs bounded by Y_t. We define the following notation to support the RL reward function:

TABLE 2. Reward function notation summary.

Notation	Meaning
ϕ	The desired classification probability on the polytope boundaries.
f	Number of non-empty homogeneous polytopes Y_t within set Y.
$\psi_{h_i}(G)$	The nearest point on $h_i(x)$ to G.
$\eta_i(G)$	The point $\psi_{h_i}(G) + \epsilon \cdot u$, where u is a unit vector randomly chosen from the hyperplane $h_i(x)$
$\Pi_i(G)$	The probability of classification $\hat{C}$ at $\eta_i(G)$, as calculated by NFGnn.
$\mathcal{H}_m(G)$	The set of the m closest hyperplanes, $h_i(x)$, to G.
$\xi_i(G)$	A positional reward that scores the location of a point in the classification probability space.
$\tau_{h_i}(G)$	The Euclidean distance $\|\|\psi_{h_i}(G), G\|\|$.
$\theta(G)$	Function defining the quality of hyperplane placement given the classification probability $\Pi(G)$.
n_t	The weighted value of an NFGs in homogeneous polytope Y_t.
S_t	The reinforcement learning reward function at time increment t.

Definition 3: Let $\psi_{h_i}(G)$ be the nearest point on $h_i(x)$ to G, and let $\eta_i(G) = \psi_{h_i}(G) + \epsilon \cdot u$, where u is a unit vector randomly chosen from the hyperplane $h_i(x)$.

Let $\Pi_i(G)$ be the probability of classification $\hat{C}$ at $\eta_i(G)$, where G in the interior of Y_t and $h_i(x)$ is a hyperplane defining some of the bounds of Y_t.

Let $\mathcal{H}_m(G) = \{h_i(x) \mid h_i(x)$ is among the m closest hyperplanes to $G\}$.

Let $\xi_i(G) = e^{-\frac{(\Pi_i(G)-\phi)^2}{2c^2}}$ be a positional reward that scores the location of a point in the classification probability space, where ϕ is the desired probability on the polytope boundaries, i.e., 0.5.

Let $\tau_{h_i}(G)$ be the Euclidean distance $||\psi_{h_i}(G), G||$. This parameter incentivizes the RL to prefer tighter polytope boundaries.

Let $\theta(G)$ define an NFG's hyperplane position and orientation quality function with respect to G:

$$\theta(G) = \sum_{h_j \in \mathcal{H}_m(G)} \sum_{i=1}^{m} \frac{\xi_i(G)}{\tau_{h_i}(G)^2}$$

with tuning factor c experimentally set to 0.1.

Let n_t define the weighted value of the NFGs inside homogeneous polytope $Y_t \in Y$:

$$n_t = \sum_{G \in Y_t} \theta(G)$$

Finally, the reward function S_t is:

$$S_t = \sum_{t=1}^{f} \frac{n_t}{f \cdot w}$$

The intuition behind the reward function is as follows:

1) Encouraging Homogeneous Polytopes: The function rewards the creation of polytopes that encompass multiple NFGs, so long as reasonable boundaries are set. The main term influencing this outcome is the denominator $f \cdot w$ which encourages a higher score per NFG per homogeneous polytope.
2) Aligning with Classification Boundaries: Through the $\xi_i(G)$ component, the function incentivizes placing polytope boundaries close to the 50% classification probability contours, ensuring accurate capture of the underlying classification landscape.
3) Orienting with the Classification Space: The summation in $\theta(G)$ over points, $\eta_i(G)$, contributes to hyperplane orientation by scoring a region of samples on each hyperplane surface. Hyperplanes that are misaligned with the probability space will be punished by $\xi_i(G)$'s exponential function.
4) Promoting Tighter Boundaries: The division by $\tau_{h_i}(G)$ encourages tighter polytope boundaries, creating more compact representations. Where $\xi_i(G)$ finds favorable locations for polytopes, $\tau_{h_i}(G)$ ensures that these locations that tightly bound their interior NFGs.

By combining these elements, S_t guides the RL algorithm towards solutions that not only meet the classification criteria but also maximize the spatial and structural efficiency of the polytopes. The function's design ensures a balance between accuracy (alignment with classification boundaries) and simplicity (larger, fewer polytopes) in the final representation.

The Complexity of the scoring function is $O(wm^2\sigma)$. The scoring function runs over all non-empty polytopes $1 \leq t \leq f$ that in the worst case can be the number of NFGs w. I.e., each polytope is non-empty and each polytope contains a single NFG. For each non-empty polytope the scoring function computes n_t which in turn computes for each of the m closest hyperplanes to G, m times the positional rewards. The positional rewards are based on the calculation of the NFGnn, which is a constant time calculation, and the Euclidean distance between the nearest point on the hyperplane to the NFG at hand. Finding each such nearest point is computed by a LP computation. As of this writing the best running time for such an algorithm [34] is $\sigma = O((w^{\mu} + w^{2.5-\iota/2} + w^{2+1/6}) \log w$ where μ is the exponent of matrix multiplication ($\mu \leq 2.38$) and ι is the dual exponent of matrix multiplication ($\iota \geq 0.31$).

2) MIXED-CLASSIFICATION POLYTOPES

After the initial RL algorithm is called, we recursively apply the reinforcement-learning algorithm to each mixed-classification polytope as illustrated in Fig. 3.

Fig. 5 demonstrates space with mixed-classification polytopes. Polytopes $\bar{k_1}k_2\bar{k_3}$ and $k_1k_2\bar{k_3}$ isolate two instances of C_2 and one instance of C_1, which means that we have mixed-classification and a recursive call for RL algorithm is performed as is illustrated in Fig. 3.

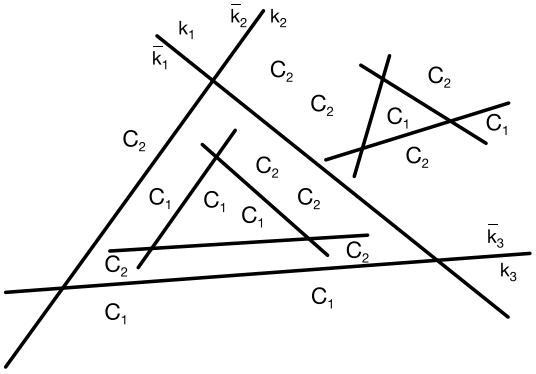


FIGURE 3. RL recursive call on mixed-classified polytopes.

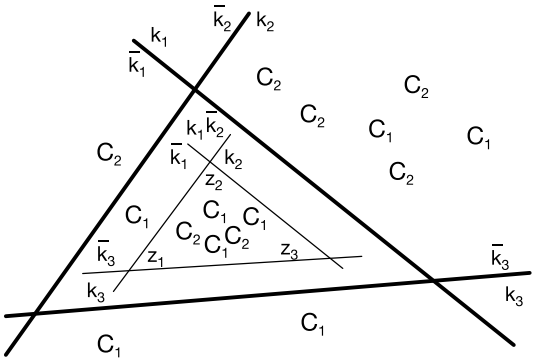


FIGURE 4. At some RL separation iteration, hyperplanes could fail to split instances, and the process will not advance.

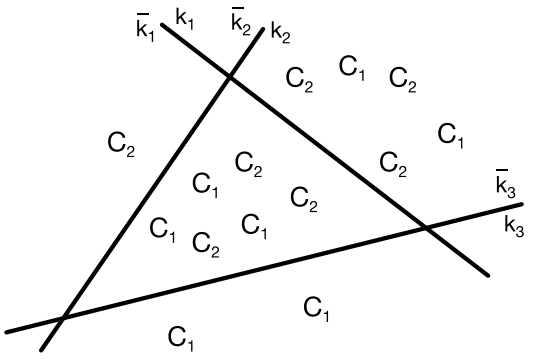


FIGURE 5. Space with mixed-classification polytopes.

The RL polytope bounding algorithm follows:

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 Fig. 4 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.

Algorithm 1 RL Bound Regions

procedure RL Bound Regions(P)
Input:
P, the vectorized NFG matrices and hyperplanes
Output:
Y_{homog}, a set of homogeneously classified polytopes
Y_{mixed}, a set of mixed classification polytopes

Let Y_{homog} be a set of homogeneously-classified polytopes
Let Y_{mixed} be a set of mixed-classification polytopes
Let Y_{active} be the working set of polytopes
Call the RL algorithm to create polytopes assigned to Y_{active}
for y in Y_{active} **do**
if y is homogeneously classified **then**
add y to Y_{homog}
else
if y is mixed classified **then**
Call RL Bound Regions with a copy of y
if RL Bound Regions returned any homogenous polytopes **then**
Add the new homogeneous regions to Y_{homog}
if RL Bound Regions also returned some mixed regions **then**
Add the mixed regions to Y_{active}
else
There was no change, so add y to Y_{mixed} for resolution with the separation algorithm
return Y_{homog} and Y_{mixed}

3) 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 Fig. 4 results in Fig. 6(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 follows:

B. NFG REMAPPING

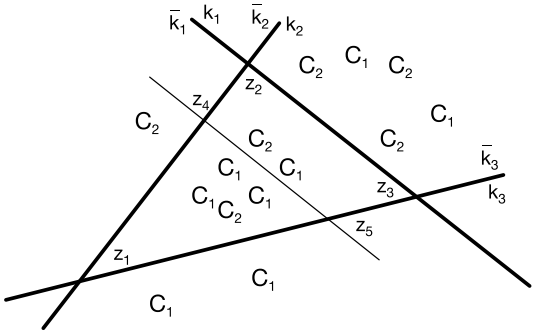
Given an input NFG matrix, G, we want to find the minimal modification that will cause players to play the actions defined by classification $\hat{C}$. We remap G to play $\hat{C}$ by locating the nearest polytope that isolates NFGs classified as $\hat{C}$ and then find the closest point on that polytope (Fig. 6(b)), G'. To increase the probability that the modified NFG will play $\hat{C}$, G' is moved toward the point with the steepest probability 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,[4] and thus the distance between each polytope and G.

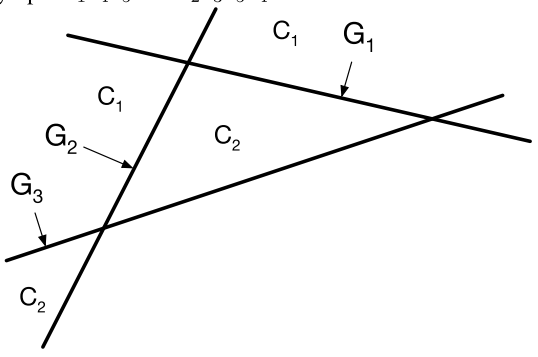
Algorithm 2 Separation Algorithm

procedure Separate(Y_{mixed})
Input:
Y_{mixed}, a set of mixed-classification polytopes
Output:
Y_{homog}, a set of homogeneously-classified polytopes

$Y_{homog} = \{\}$
for Y in Y_{mixed} **do**
Generate a hyperplane h to separate any two NFGs in Y
Polytopes Y_1 and Y_2 result from Y's separation by h
if Y_1 is homogenously-classified **then**
Add Y_1 to Y_{homog}
else if Y_1 is missclassified **then**
Call Separate(Y_1.copy())
Add result to Y_{homog}
if Y_2 is homogenously-classified **then**
Add Y_2 to Y_{homog}
else if Y_2 is missclassified **then**
Call Separate(Y_2.copy())
Add result to Y_{homog}
return Y_{homog}



(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 6. Separation algorithm illustrations.

[4] In the case where G is already in a relevant polytope, then the algorithm returns 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 $(X_1, \ldots, X_m)$ and a set of k hyperplanes defines a matrix with k where the first row is $\Sigma_{j=1}^{m} b_{1j} * x_j = d_1$ and the kth row is $\Sigma_{j=1}^{m} b_{kj} * x_j = d_k$.

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 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. In Fig. 7(a) and Fig. 7(b) we give an example of how to remap from G to G'. 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 $(X_1'', \ldots, X_m'')$, and move G' toward $(X_1'', \ldots, X_m'')$ 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. Fig. 7(b) illustrates this operation.

Fig. 7(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:

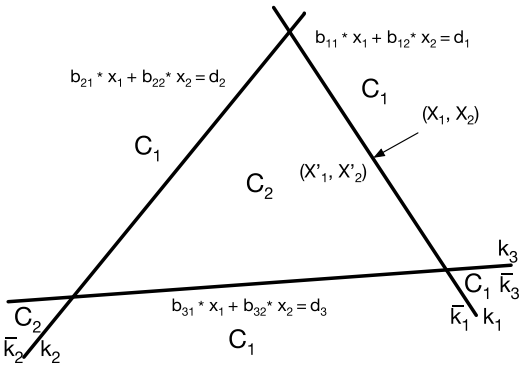
$$b_{11} * x_1 + b_{12} * x_2 < d_1$$
$$b_{21} * x_1 + b_{22} * x_2 < d_2$$
$$b_{31} * x_1 + b_{32} * x_2 < d_3$$

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

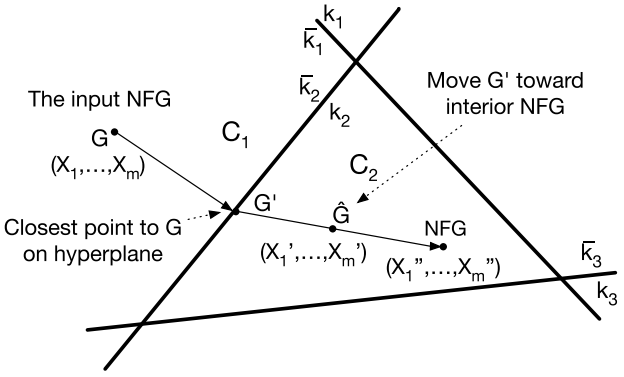
$$b_{11} * x_1 + b_{12} * x_2 < d_1$$
$$b_{21} * x_1 + b_{22} * x_2 > d_2$$
$$b_{31} * x_1 + b_{32} * x_2 > d_3$$

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

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.



(a) Finding the nearest point on a polytope surface



(b) Improving Remapping Confidence

FIGURE 7. NFG remapping illustrations.

C. THE ALGORITHM

Algorithm 3 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 classifications 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 classification $\hat{C}$

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 (Algorithm 1) 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 with the steepest probability gradient in $\hat{Y}_t$.
Call the new location $\hat{G}$.
return $\hat{G}$

IV. EXPERIMENTS

This section presents an experiment to evaluate the NFG-remapping system. Since no comparable benchmark mechanisms exist, we test GUIDE's output versus a panel of classifiers. The experiments also 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. The panel consists of known classifiers trained on our dataset of played NFGs. Appendix -C provides details on the classier panel.

A. EXPERIMENT DESIGN

We trained GUIDE using datasets from the combined observations of several experimental game theory studies in which participants were paid to play NFGs. 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.

Sixty-one, 3×3, 2-player NFGs were extracted from [3], [5], [6], [25], [26], [27], [28], [29], and [30]. These games were played 4,812 times overall. However, many outcomes were identical, which resulted in a distribution of outcomes for a smaller number of games. The availability of human data, not a limitation of GUIDE, dictated the use of 3×3 matrices. The result examines the action selected by the row player, a choice made for clarity despite the possibility of examining a combination of actions. The available actions were named actions 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. 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. 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.

The system was evaluated with 2000 randomly generated test NFGs to be remapped. Prior to evaluation, the system was trained with 71 NFGs taken from the experimental data. The NFGs were classified using their most probable outcome, per GUIDE's mechanism. As the system performed well with sparse actual data, there was no need to generate synthetic training data to assist GUIDE.

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

1) We selected several ML classifiers: SVM [10], Random Forest [12], [35], Multi-layer Perceptron [13], and NFGnn [3].
2) The classifiers were trained on the original 4,812 game matrix instances from [3], [5], [6], [25], [26], [27], [28], [29], and [30].
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.

GUIDE converts the dataset's 3×3 two-player NFGs into a *18-dimensional* vectors and then separates those NFGs into homogeneously-classified polytopes.

As the RL algorithm will decide whether and how to use the available hyperplanes, the initial hyperplane count, k is not critical to success so long as sufficient hyperplanes are provided. We chose to start with the number of distinct NFSs minus one. This is the number of hyperplanes needed to bound each distinct NFG. Overfitting is not a problem given that GUIDE's goal is to clearly define the boundaries as defined by another properly trained classifier, in this case NFGnn. Tightly fitting the classification boundaries is in fact the objective.

B. EXPERIMENT RESULTS

The experimental results demonstrated high agreement with the classifier panel. Particularly noteworthy is the agreement at the initial boundaries. The agreement at the boundaries indicates the quality of the polytopes constructed by the RL. We were pleased to see that the initial agreement with NFGnn was slightly higher than 50%, indicating that the ensemble RL technique successfully influenced the placement of the bounding hyperplanes to fall on or near the 50% probability boundaries. As expected, classifier agreement approaches 100% as the sample points progressed along the steepest probability gradient within the target polytopes.

The level of agreement in the polytope interiors is subject to the strength of the desired classification in the target polytopes as well as the idiosyncrasies of each classifier on the panel. Though classifier agreement increase as sampled points follow the steepest probability gradient, the strength of the desired classification in the target polytope affects the uptake in agreement. The interior samples also reflect the consistency of each classifier in the space of the target polytopes. Even so, the fact that interior agreements are monotonically increasing is a good indication that the polytopes are correctly sized and are not inadvertently capturing subregions with a classification other than the target classification.

The detailed experimental results and the classifier panel are presented in tables in appendix -C and -D. The tables are summarized below in Fig. 8, which also provides whiskers for the 95% Wilson confidence interval [36] for each datapoint. We chose the Wilson interval for its accuracy with small samples and relative indifference to the idiosyncrasies of the various classifiers on the panel.

In our experiment, the null hypothesis (H0) posits that the variance in the system's agreement with NFGnn is no

different from the variance observed in a series of coin tosses, both assumed to center around a 50% success rate. The alternative hypothesis (H1), however, suggests that the system's agreement with NFGnn is more consistently centered around this 50% mark, displaying less variance compared to random coin tosses.

Levene's test [37] is well suited for our scenario because it specifically assesses variances rather than means. While both the system and the coin tosses might average a 50% success rate, our hypotheses focus on the consistency of achieving this rate, which is fundamentally a question of variance. Levene's test, known for its robustness against non-normality, is ideal for determining if the differences in variance between the two distributions are statistically significant, without being misled by any deviations from a normal distribution that might otherwise affect the test's accuracy.

In this case, a p-value of 0.017 (less than 0.05 is typically considered significant) supports the alternative hypothesis (H1). This low p-value indicates that the variance in the system's results is statistically significantly less than that of the coin tosses, confirming that the system achieves a 50% success rate more consistently than would be expected by random chance.

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. Fig. 8 summarizes this relationship and shows that the system can be configured for a preferred accuracy. One can see how 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.

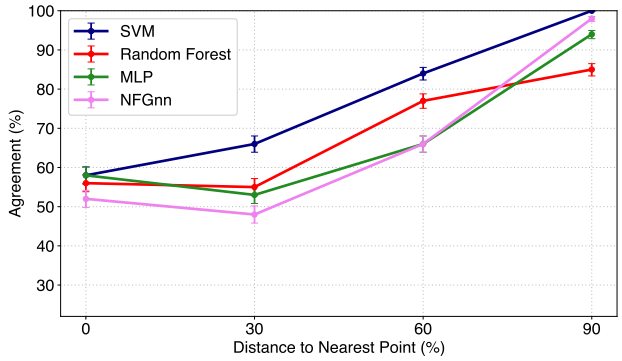


FIGURE 8. Classifier agreement on $\hat{C}$ vs % distance to nearest point.

V. CONCLUSION AND DISCUSSION

We introduced GUIDE, an ensemble reinforcement-learning-based mechanism for helping game designers create games with desirable outcomes. Human players played the influenced games, so players did not always play the equilibrium strategy. The mechanism is general and applies to other game theory problems; see appendix -F. The experimental results showed that the system operated as expected and was reliable relative to the performance of the testing classifiers.

A. NFG DISTANCE

Consider the NFG in Fig. 10, a matching pennies variant with a modified payoff for each player. The game has two players with "Heads" or "Tails" actions. Analysis reveals a single mixed Nash equilibrium: the row player plays Heads with probability $p = 2/5$ and Tails with probability $1 - p = 3/5$; the column player plays Heads with probability $q = 1/3$ and Tails with probability $1 - q = 2/3$. The most frequently played class is the joint action Tails (row player), Tails (column player), which classifies the game such that player 1 wins and player 2 loses.

The designer that might be concerned that the second player will leave the game[5] or for some other reason, wants to change the game such that the most frequently played class is the joint action Heads (row player), Tails (column player). Assuming the designer has a space of classified NFGs and the NFG in Fig. 9 is located in one of the classes with classification Heads (row player), Tails (column player). The distance between the NFG in Fig. 10 and the NFG in Fig. 9 is 2.84. The change in potential payoffs that the designer has to obligate for is 1 less in one component and 2 and 2/3 more in another component, overall 1 and 2/3. Now the designer can influence the original NFG from Fig. 9 to turn into an NFG in the desired classification of Heads (row player), Tails (column player) by finding a closer NFG within the same classification space. Such NFG can be found in Fig. 11. The distance between the NFG in Fig. 10 and the NFG in Fig. 11 is 1.536. The change in potential payoffs that the designer has to obligate for is 1 less in one component and 1 and 1/6 more in another component. I.e., overall 1/6.

	$q = 0.4$	$1 - q = 0.6$
$p = 0.667$	2,-1	-1,1
$p = 0.333$	-1,3	1,-1

FIGURE 9. Game 1.

	$q = 1/3$	$1 - q = 2/3$
$p = 2/5$	3,-1	-1,1
$1 - p = 3/5$	-1,1/3	1,-1

FIGURE 10. Game 2.

	$q = 0.4$	$1 - q = 0.6$
$p = 0.555$	2,-1	-1,1
$1 - p = 0.444$	-1,1.5	1,-1

FIGURE 11. Game 3.

[5]Calculating the expected utilities of the two players in this equilibrium gives us $u_1 = 1/3$ for the row player, and $u_2 = -1/5$ for the column player.

B. CONSTRUCTING M, H AND D

1) EXAMPLE FOR CONSTRUCTING *M*

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.

2) EXAMPLE FOR CONSTRUCTING *H*

Given $k = 5$ hyperplanes and NFGs coefficients of dimension $m = 8$, we can describe matrix H in the following way:

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

3) EXAMPLE FOR CONSTRUCTING *D*

Given $k = 5$ hyperplanes and $w = 5$ NFGs transformed to vectors, we can describe the 5×5 matrix D in the following way:

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

C. THE CLASSIFIER PANEL

We selected several standard ML classifiers for the panel. The classifiers are SVM, Random Forest, Multi-layer Perceptrons, and NFGnn. NFGnn [3] is a state-of-the-art neural network designed to predict the action played by NFGs. NFGnn performs well in this domain compared to other classifiers.

Selected classifiers were trained on the original 4,812 game data instances, which were divided into a 2/3 training set and a 1/3 test set. SVM and Random Forest were configured with parameter class_weight = balanced, and MLP was configured with three hidden layers. All classifiers were trained for 10,000 epochs.

The classifiers, except NFGnn, were not particularly accurate, which was also noted in [3]. The Classifier Accuracy Benchmark results in Table 3 compare the accuracy of the Random Forest, Multi-layer Perceptron Classifier, SVM, and NFGnn trained with 800 epochs.

NFGnn required special handling as it outputs a conditional distribution of human play rather than a specific classification for a given NFG. We handle this by setting NFGnn's prediction to the classification it gave with the highest probability.

TABLE 3. Classifier accuracy benchmark (%).

Action	SVM	RF	MLP	NFGnn
A	67	65	69	100
B	66	66	66	66
C	54	56	53	100
Overall	63	63	64	88

D. DETAILED EXPERIMENTAL RESULTS

This section provides a summary of the findings related to Fig. 8, focusing on the agreement between GUIDE and various classifiers in terms of the selected action and the percentage of distance covered toward the nearest classified point within the containing polytope. The concordances are detailed in Table 4 for the panel's SVM, Table 5 for the Random Forest, Table 6 for the MLP, and Table 7 for the NFGnn.

TABLE 4. SVM agreement with $\hat{C}$ (%).

Action	Distance to Nearest Point (%)			
	0	0.3	0.6	0.9
A	90	68	96	100
B	32	57	69	100
C	53	74	93	100
Overall	58	66	84	100

TABLE 5. Random forest agreement with $\hat{C}$ (%).

Action	Distance to Nearest Point (%)			
	0	0.3	0.6	0.9
A	82	23	81	100
B	23	54	56	55
C	63	87	96	100
Overall	56	55	78	85

TABLE 6. MLP agreement with $\hat{C}$ (%).

Action	Distance to Nearest Point (%)			
	0	0.3	0.6	0.9
A	46	36	62	100
B	64	64	70	94
C	64	58	68	100
Overall	58	53	67	94

TABLE 7. NFGnn agreement with $\hat{C}$ (%).

Action	Distance to Nearest Point (%)			
	0	0.3	0.6	0.9
A	75	56	72	97
B	37	47	68.5	100
C	45	42	59	98
Overall	52	48	64	98

E. EXPERIMENTAL RESULTS FOR NON-NFG DATASET

The Iris [38] dataset was used to evaluate how the system would perform when operating on datasets not based on

NFGs. Each randomly generated test point was remapped to all possible classifications, and agreement with a panel of classifiers was tested to estimate the system's performance. Since [38] does not contain NFGs, the NFGnn classifier was not included in the panel for these datasets.

F. GUIDE FOR GENERAL USE

GUIDE was designed and developed to apply to a wide range of problems that can be represented as classification problems. To evaluate performance with non-NFG data, we executed tests on the Iris [38] dataset. In these tests GUIDE ran without an ensemble reward function by setting $\theta(G) = 1$.

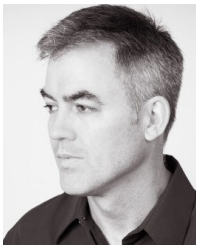
We used ∼85% of the Iris dataset [38]'s 150 data points to train our mechanism and kept the remaining ∼15% for testing. We used the same methodology used in the experimental results section. Results are provided for 70 hyperplanes, 10 episodes, and 2000 steps each (total 20,000 steps). The agreement between GUIDE's classifier and the panel is presented in Table 8.

TABLE 8. Iris dataset classifier agreement (%).

		Distance to Nearest Point (%)				
		0.1	0.3	0.5	0.7	0.9
Classifier	SVM	57	71	76	77	97
	Random Forest	55	63	70	74	75
	MLP	42	61	67	75	90

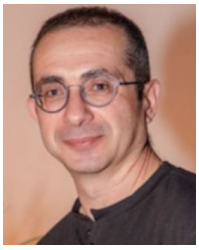
REFERENCES

[1] T. Sandholm, "Automated mechanism design with a structured outcome space," in *Principles and Practice of Constraint Programming—CP 2003*, F. Rossi, Ed., Berlin, Germany: Springer, 2003, pp. 19–36.
[2] B. Lucier and R. P. Leme, "GSP auctions with correlated types," in *Proc. 12th ACM Conf. Electron. Commerce*, New York, NY, USA, Jun. 2011, pp. 71–80.
[3] J. Hartford, J. Wright, and K. Leyton-Brown, "Deep learning for predicting human strategic behavior," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2424–2432.
[4] J. R. Wright and K. Leyton-Brown, "Level-0 models for predicting human behavior in games," *J. Artif. Intell. Res.*, vol. 64, pp. 357–383, Feb. 2019.
[5] D. Fudenberg and A. Liang, "Predicting and understanding initial play," *Amer. Econ. Rev.*, vol. 109, no. 12, pp. 4112–4141, Dec. 2019.
[6] D. Cooper and J. Van Huyck, "A cognitive hierarchy model of games," *J. Econ. Theory*, vol. 110, no. 2, pp. 290–308, 2003.
[7] J. R. Wright and K. Leyton-Brown, "Level-0 meta-models for predicting human behavior in games," in *Proc. 15th ACM Conf. Econ. Comput.*, Jun. 2014, pp. 857–874.
[8] J. R. Wright and K. Leyton-Brown, "Beyond equilibrium: Predicting human behavior in normal-form games," in *Proc. 24th AAAI Conf. Artif. Intell.*, 2010, pp. 901–907.
[9] B. Bošanský, V. Lisý, M. Lanctot, J. Cermák, and M. H. M. Winands, "Algorithms for computing strategies in two-player simultaneous move games," *Artif. Intell.*, vol. 237, pp. 1–40, Aug. 2016.
[10] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sep. 1995.
[11] D. Rumelhart, G. Hinton, and R. Williams, *Learning Internal Representations By Error Propagation*. Cambridge, MA, USA: MIT Press, 1986, pp. 318–362.
[12] Y. LeCun, L. Bottou, G. Orr, and K.-R. Müller, *Efficient BackProp*. Berlin, Germany: Springer, 2012, pp. 9–48.
[13] T. Kam Ho, "Random decision forests," in *Proc. 3rd Int. Conf. Document Anal. Recognit.*, vol. 1, 1995, pp. 278–282.
[14] M. George and S. Kamara, "Adversarial level agreements for two-party protocols," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, May 2022, pp. 816–830.
[15] N. I. Schwartzbach, "Payment schemes from limited information with applications in distributed computing," in *Proc. 23rd ACM Conf. Econ. Comput.*, Jul. 2022, pp. 129–149.
[16] D. Balduzzi, S. Racaniere, J. Martens, J. Foerster, K. Tuyls, and T. Graepel, "The mechanics of n-player differentiable games," in *Proc. 35th ICML*, 2018, pp. 354–363.
[17] M. Al-Shedivat, T. Bansal, Y. Burda, I. Sutskever, I. Mordatch, and P. Abbeel, "Continuous adaptation via meta-learning in nonstationary and competitive environments," 2017, *arXiv:1710.03641*.
[18] D. Hadfield-Menell, S. Russell, P. Abbeel, and A. Dragan, "Cooperative inverse reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 1–9.
[19] Y. Vorobeychik, C. Kiekintveld, and M. P. Wellman, "Empirical mechanism design: Methods, with application to a supply-chain scenario," in *Proc. 7th ACM Conf. Electron. Commerce*, Jun. 2006, pp. 306–315.
[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.
[21] D. H. Wolpert, "Stacked generalization," *Neural Netw.*, vol. 5, no. 2, pp. 241–259, Jan. 1992. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608005800231
[22] Y. Yu and M. Yao, "When convolutional neural networks meet laser-induced breakdown spectroscopy: End-to-end quantitative analysis modeling of ChemCam spectral data for major elements based on ensemble convolutional neural networks," *Remote Sens.*, vol. 15, no. 13, p. 3422, Jul. 2023. [Online]. Available: https://www.mdpi.com/2072-4292/15/13/3422
[23] J. Du, B. Jiang, C. Jiang, Y. Shi, and Z. Han, "Gradient and channel aware dynamic scheduling for over-the-air computation in federated edge learning systems," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 4, pp. 1035–1050, Apr. 2023.
[24] S. K. Pal, S. Bandyopadhyay, and C. A. Murthy, "Genetic algorithms for generation of class boundaries," *IEEE Trans. Syst., Man, Cybern., B*, vol. 28, no. 6, pp. 816–828, Dec. 1998.
[25] C. Camerer, T. Ho, and J. Chong, "A cognitive hierarchy model of games," *Quart. J. Econ.*, vol. 119, no. 3, pp. 861–898, Aug. 2004.
[26] E. Haruvy, D. O. Stahl, and P. W. Wilson, "Modeling and testing for heterogeneity in observed strategic behavior," *Rev. Econ. Statist.*, vol. 83, no. 1, pp. 146–157, Feb. 2001.
[27] M. Plappert. *Keras-R*. Accessed: Feb. 16, 2022. [Online]. Available: https://github.com/keras-rl/keras-rl
[28] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[29] D. O. Stahl and P. W. Wilson, "Experimental evidence on players' models of other players," *J. Econ. Behav. Org.*, vol. 25, no. 3, pp. 309–327, Dec. 1994.
[30] D. O. Stahl and P. W. Wilson, "On players' models of other players: Theory and experimental evidence," *Games Econ. Behav.*, vol. 10, no. 1, pp. 218–254, 1995.
[31] L. Floridi, J. Cowls, M. Beltrametti, R. Chatila, P. Chazerand, V. Dignum, C. Luetge, R. Madelin, U. Pagallo, F. Rossi, B. Schafer, P. Valcke, and E. Vayena, "AI4People—An ethical framework for a good AI society: Opportunities, risks, principles, and recommendations," *Minds Mach.*, vol. 28, no. 4, pp. 689–707, Dec. 2018.
[32] S. Barocas, M. Hardt, and A. Narayanan, *Fairness and Machine Learning: Limitations and Opportunities* (Adaptive Computation and Machine Learning series). MIT Press, 2023. [Online]. Available: https://books.google.co.il/books?id=HuGwEAAAQBAJ
[33] M. Spizel, *Vector Analysis*. New York, NY, USA: McGraw-Hill, 1959.
[34] Y. Tat Lee, Z. Song, and Q. Zhang, "Solving empirical risk minimization in the current matrix multiplication time," 2019, *arXiv:1905.04447*.
[35] D. E. Rumelhart and J. L. McClelland, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, vol. 1, D. Rumelhart and J. McClelland, Ed., Cambridge, MA, USA: MIT Press, 1987, pp. 318–362.
[36] E. B. Wilson, "Probable inference, the law of succession, and statistical inference," *J. Amer. Stat. Assoc.*, vol. 22, no. 158, pp. 209–212, Jun. 1927.
[37] H. Levene, "Robust tests for equality of variances," in *Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling*, I. Olkin, Es., Palo Alto, CA, USA: Stanford Univ. Press, 1960, pp. 278–292.
[38] R. A. Fisher, "Use of multiple measurements in taxonomic problems," *Ann. Eugenics*, vol. 7, no. 2, pp. 179–188, Sep. 1936.



C. BRADY received the B.Sc. degree in computer science from the University of California, Berkeley, USA. He is currently pursuing the M.Sc. degree. He holds patents in machine learning, electronic commerce, and security. His research interests include computational game theory, machine learning, and artificial intelligence.

R. GONEN received the M.Sc. and Ph.D. degrees in computer science from Hebrew University, in 2001 and 2006, respectively. She was a Postdoctoral Fellow with Bell Labs and held a research position with Yahoo Labs, California. She is currently an Associate Professor with the Department of Mathematics and Computer Science, The Open University of Israel. Her research interests include computational social choice, combinatorial auctions and markets, social networks, and computational game theory. The mechanisms she designed have been implemented in industry and have led to cost savings of over a hundred million dollars in the supply chain of a major American company. She serves as a senior program committee member and a program committee member for the leading AI conferences. She received research grants from European Union and Israeli Ministry of Science and has also developed over a dozen international patents in AI.



G. RABINOVICH received the B.Sc. degree in computer science from Ben Gurion University, Israel, in 2001. He is currently pursuing the M.Sc. degree in computer science. He has been with hi-tech industry in ML/AI, since 2001. His research interests include machine learning and artificial intelligence.