Incremental Multiple Oracle

Carlos Martin
Carnegie Mellon University
cgmartin@cs.cmu.edu

Tuomas Sandholm
Carnegie Mellon University
Strategy Robot, Inc.
Optimized Markets, Inc.
Strategic Machine, Inc.
sandholm@cs.cmu.edu

Abstract

We present a framework for computing approximate mixed-strategy Nash equilibria of continuous-action games. It is a modification of the multiple oracle algorithm, which extends the double oracle algorithm to multiple players and continuous action spaces. Unlike prior oracle-based methods, it maintains fixed-cardinality pure strategy sets for each player. Thus, its memory requirements do not grow with the number of iterations. Furthermore, it does not require exact metagame solving on each iteration, which can be computationally expensive for large metagames. Moreover, it does not require global best-response computation on each iteration, which can be computationally expensive or even intractable for high-dimensional action spaces and general games. Our method gradually reduces the exploitability of the strategy profile in the finite metagame, pushing it toward Nash equilibrium. Simultaneously, it gradually improves the pure strategies against this strategy profile in the full game. We evaluate our method on various continuous-action games, showing that it obtains approximate mixed-strategy Nash equilibria with low exploitability.

1 Introduction

Research on computing *Nash equilibria (NE)* in games has mostly focused on settings with finite, discrete action spaces. However, many games involving space, money, or time have continuous action spaces. These include security games in continuous spaces [53–55], resource allocation games [36], network games [38], simulations of military scenarios and wargaming [67], and video games [11, 115]. Also, even if a game is discrete, it may be fine-grained enough to treat as continuous in order to improve the computational efficiency of equilibrium finding; in poker, for example, this has been done for bet sizes [22] and for private hands [24, 37]. The usual approach to computing an equilibrium of a game with continuous action spaces involves discretizing the action space. That entails a loss in solution quality [57] and does not scale well to high-dimensional spaces. Therefore, other approaches are called for.

We present a new algorithm, *Incremental Multiple Oracle (IMO)*, for computing approximate mixed-strategy NE in continuous-action games. It is a modification of the multiple oracle algorithm [59], which extends the double oracle algorithm [79] to multiple players and continuous action spaces. The method gradually reduces the exploitability of the strategy profile in the finite metagame, pushing it toward NE. Simultaneously, it gradually improves the pure strategies against this strategy profile in the full game. We evaluate the method on various continuous-action games, showing that it obtains approximate mixed-strategy NE with low exploitability. Unlike prior oracle-based methods, this method runs in constant memory in the number of iterations, because it maintains fixed-cardinality pure strategy sets for each player. Furthermore, it does not require exact metagame solving on each iteration, which can be computationally expensive for large metagames. Moreover, it does not require

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

global best-response computation on each iteration, which can be computationally expensive or intractable for general games.

This paper is structured as follows. In §2, we introduce relevant notation and present a mathematical formulation of the problem we are tackling. In §3, we describe related work. In §4, we present our method. In §5, we describe the experimental settings we use as benchmarks, and present our experimental results for them. In §6, we present our conclusions and suggest directions for future research. In the appendix, we present a theoretical analysis (Appendix A), additional experimental results, including direct comparisons with prior methods and ablations (Appendices B–D), additional related research (Appendix E), and example code (Appendix F).

2 Problem Formulation

We use the following notation. Between vectors or matrices, $\odot$ denotes the element-wise (i.e., Hadamard) product. If $n \in \mathbb{N}$, $[n] = \{0, \dots, n-1\}$. If S is a set, $\triangle S$ is the set of all Borel probability measures on S. If f is a function, then $f[x \mapsto y]$ is the new function defined by $f[x \mapsto y](x') = y$ if $x' = x$ and $f(x')$ otherwise. This is a standard notation for *substitution*.[1] If $\{S_i\}_{i \in I}$ is a family of sets, then $\Pi S = \prod_{i \in I} S_i$ is their Cartesian product. If $\{\mu_i\}_{i \in I}$ is a family of measures, then $\bigotimes_{i \in I} \mu_i$ is their product measure. If $i \in \mathbb{N}$, then $\mathbf{e}_i$ is the ith standard basis (one-hot) vector. The softmax function is defined by $\operatorname{softmax}(\mathbf{x})_i = \frac{\exp \mathbf{x}_i}{\sum_{j \in [n]} \exp \mathbf{x}_j}$ for $\mathbf{x} \in \mathbb{R}^n$.

A *strategic-form game* is a tuple (I, S, u) where I is a set of players, S_i is a set of strategies for each player $i \in I$, and $u_i : \Pi S \to \mathbb{R}$ is a utility function for each player $i \in I$. Here, $\Pi S = \prod_{i \in I} S_i$ is the set of strategy profiles. Given a strategy profile, a *best response (BR)* for a player is a strategy that maximizes its utility given the other players' strategies. That is, given a strategy profile $s \in \Pi S$, a BR for player $i \in I$ is an element of the best-response set $\mathrm{BR}_i(s) = \operatorname{argmax}_{x \in S_i} u_i(s[i \mapsto x])$. An NE is a strategy profile for which each player's strategy is a BR to the other players' strategies. That is, it is a strategy profile $s \in \Pi S$ such that $s_i \in \mathrm{BR}_i(s)$ for every player $i \in I$.

Given a strategy profile $s \in \Pi S$, player i's individual regret is $r_i(s) = \sup_{x \in S_i} u_i(s[i \mapsto x]) - u_i(s)$, which is the highest utility that player could gain by unilaterally changing its strategy. Let $R = \sup_{i \in I} r_i$ be the maximum regret across players. An ε-NE is a strategy profile $s \in \Pi S$ such that $R(s) \leq \varepsilon$. Likewise, the *exploitability* is defined as $E = \sum_{i \in I} r_i$, i.e., the sum of player regrets (also called *NashConv* [61]). Then R and E are bounded in terms of each other as follows: $E/|I| \leq R \leq E \leq R|I|$. Furthermore, they are nonnegative and zero precisely at NE. Therefore, if NE exist, finding them is equivalent to minimizing R or E [77, p. 91, eq. 2.9]. Consequently, R and E are standard measures of "closeness" to NE in the literature [61, 65, 117, 113].

For any game (I, S, u), there exists a mixed-strategy game $(I, \Sigma, \bar{u})$ where $\Sigma_i = \triangle S_i$ and $\bar{u}_i(\sigma) = \mathrm{E}_{s \sim \bigotimes_{j \in I} \sigma_j} u_i(s)$. That is, each player's strategy is a probability measure over its original strategy set (this is called a *mixed strategy*), and its utility is the resulting expected utility in the original game. A *mixed-strategy NE (MSNE)* is an NE of the mixed-strategy game.[2]

A continuous-action game is a game whose strategy sets are subsets of Euclidean space, e.g., $S_i \subseteq \mathbb{R}^d$. The existence and uniqueness of NE for such games have been studied extensively in the literature. We describe some of these results in Appendix E. Most equilibrium-finding algorithms in the literature target discrete-action games, raising the question of how to compute equilibria for continuous-action games.

3 Related Research

McMahan et al. [79] introduced *Double Oracle (DO)*, an algorithm that computes NE of two-player zero-sum normal-form games. It starts with an initial set of pure strategies[3] for each player, called

[1] In our game-theoretic context, we use the notation $u_i(s[i \mapsto s'_i])$ instead of $u_i(s'_i, s_{-i})$ because it also works for general arguments that, unlike s'_i, may not be indexed by i.

[2] More generally, one can consider settings where randomness is a limited resource (e.g., only a limited number of random bits are available to the agent) or where the agent can only mix between a limited number of pure strategies (i.e., its mixed strategy must be *sparse*). Alternatively, its mixed strategy may be restricted to some class of *representable* distributions, such as an explicit parametric model or an implicit density model like a generative adversarial network [43].

[3] In this context, we use "pure strategy" and "action" interchangeably.

that player's *support*. On each iteration, it computes an MSNE of the subgame induced by those supports, and then computes BRs to that MSNE, adding them to the corresponding players' supports. This process repeats until convergence. DO is commonly used to find approximate MSNE of games with large strategy spaces.

Adam et al. [1] introduced an extension of DO that computes NE of two-player zero-sum games with *continuous*, compact strategy sets. In the authors' words, "The equilibrium of each finite subgame is found by solving a linear program. The best responses were computed by selecting the best point of a uniform discretization for the one-dimensional problems and by using a mixed-integer linear programming reformulation for the Colonel Blotto games." They proved that every accumulation point of its iterates is an NE and that it finds an *approximate* equilibrium within finitely many iterations.

Kroupa and Votroubek [59] generalized this algorithm to n players, under the name *Multiple Oracle (MO)*. This algorithm is shown in Algorithm 1. It starts with nonempty finite supports for each player. On each iteration, it performs the following two steps. First, it computes an equilibrium of the finite metagame consisting of the restriction of the full game to the finite subsets (denoted by $u|_X$). Second, for each player, it computes a BR to this equilibrium *in the full game*, and adds it to the player's finite subset.

Kroupa and Votroubek [59] prove that MO finds an approximate equilibrium within finitely many iterations and that, in the Wasserstein metric, every accumulation point of its iterates is an MSNE of the full game. For some classes of utility functions, the BRs can be computed with global solvers. According to the authors, "the choice of best response oracle and the method for solving finite subgames should be fine-tuned for every particular class of games [...] our method converges fast when the dimensions of strategy spaces are small and the generated subgames are not large".

Wang and Chizat [118] introduced and analyzed a particle method for computing mixed NE of two-player zero-sum games whose pure-strategy sets are continuous, using first-order access to the payoffs. It represents each mixed strategy as a weighted set of atoms and updates both the weights and the positions of the atoms with proximal point steps, which can be viewed as discretizing an interacting Wasserstein–Fisher–Rao gradient flow implicitly in time. They proved that, under non-degeneracy assumptions, it converges exponentially fast to the exact mixed NE from any initialization that is sufficiently close to optimal.

In Appendix E, we describe additional algorithms based on *fictitious play (FP)* [21] and *policy-space response oracles (PSRO)* [61].

Sparse equilibria. Althöfer [4] proved that a two-player zero-sum m-action game has an ε-equilibrium of support size $O(\frac{\log m}{\varepsilon^2})$. Lipton et al. [64] proved that an n-player m-action game has an ε-equilibrium of support size $O(\frac{n^2 \log m}{\varepsilon^2})$. Hémon et al. [50] improved this to $O(\frac{n \log m}{\varepsilon^2})$. Babichenko et al. [7] improved this to $O(\frac{\log n + \log m}{\varepsilon^2})$. They also studied correlated and coarse correlated equilibria. Stein et al. [108] studied separable and low-rank continuous games, bounding the support sizes of equilibria in terms of the payoffs' complexity. Specifically, they defined a notion of *rank* for continuous games and showed that some equilibrium has every player mixing over a number of pure strategies bounded by an explicit function of the rank. This extends a result of Lipton et al. [64] on two-player games with low-rank payoff matrices to any finite number of players, infinite strategy sets, and more general payoff functions. On the other hand, in some continuous games, every equilibrium has uncountable support [40, 101].[4] However, these might still have small-support approximate equilibria (i.e., strategy profiles of low exploitability). Afiouni et al. [2] study k-sparse commitments in two-player games, in which one player may mix over at most k pure strategies. They present mixed-integer linear-program formulations and incremental strategy-generation algorithms.

4 Proposed Method

The MO algorithm has some limitations. First, metagame solving on each iteration can be expensive, especially when the metagame is large. Second, global BR computation on each iteration can be expensive or even intractable (e.g., for high-dimensional spaces or general utility functions). Third,

[4] Conversely, Rehbeck [100] showed that if each strategy space is a metric space with infinitely many points, then every profile of finitely supported mixed strategies is the unique NE of some separable game.

Algorithm 1 Multiple Oracle (MO).

input (I, S, u) is a game.
input $X_i \subseteq S_i$ is a finite nonempty initial support for each $i \in I$.
input $T \in \mathbb{N}$ is a number of iterations.
repeat T **times**
  $\sigma \in \text{MSNE}(u|_X)$ ▷ Compute an MSNE of the subgame
  for $i \in I$ **do**
    $s_i \in \text{BR}_i(u, \sigma)$ ▷ Compute a BR in the full game
    $X_i \leftarrow X_i \cup \{s_i\}$ ▷ Add it to the support
output σ

Algorithm 2 Incremental Multiple Oracle (IMO).

input (I, S, u) is a game.
input n_i is a support size for each $i \in I$.
input $X_i \in S_i^{n_i}$ is an initial vector of pure strategies for each $i \in I$.
input $L_i \in \mathbb{R}^{n_i}$ is an initial vector of logits for each $i \in I$.
input $\eta \in \mathbb{R}_{\geq 0}$ is a stepsize.
input $T \in \mathbb{N}$ is a number of iterations.
input $K_{\text{ED}} \in \mathbb{N}$ is a number of ED sub-iterations.
input $K_{\text{BR}} \in \mathbb{N}$ is a number of BR sub-iterations.
repeat T **times**
  $G \leftarrow \text{Metagame}(u, X)$ ▷ Compute induced metagame
  repeat K_{ED} **times** ▷ Reduce exploitability
    $g \leftarrow \nabla_L \text{Exploitability}(G, \{\text{softmax}(L_i)\}_{i \in I})$
    $L \leftarrow L - \eta g$
  $p \leftarrow \{\text{softmax}(L_i)\}_{i \in I}$
  repeat K_{BR} **times** ▷ Improve best responses
    for $i \in I$ **do**
      $g_i \leftarrow \nabla_{X_i} \operatorname{mix}_{j \in [n_i]} \text{Metagame}(u, X)_i(p[i \mapsto \mathbf{e}_j])$
    $X \leftarrow X + \eta g$
output X, L

the memory it requires is not fixed, but grows over time. To address these limitations, we introduce a new algorithm, shown in Algorithm 2.

Algorithm description. Each player i has a vector of logits L_i that parameterizes a probability distribution over its support X_i. Specifically, player i's probability distribution over its support is $p_i = \text{softmax}(L_i)$. We initialize these logits to zero, yielding a uniform probability distribution. We do this because the uniform distribution is symmetric, maximizes entropy, and is a natural starting point. Unlike MO, the algorithm maintains fixed-cardinality supports for each player, thus requiring only constant memory in the number of iterations. Furthermore, it replaces the exact or global oracles of the MO algorithm with approximate ones. On each iteration, the algorithm works as follows. First, it constructs the metagame of utilities that is induced by the current supports X of the players, which is denoted by $\text{Metagame}(u, X)$. Formally, $\text{Metagame}(u, X)_i(p) = \sum_{k \in \prod_{j \in I}[n_j]} u_i(\{X_{j,k_j}\}_{j \in I}) \prod_{j \in I} p_{j,k_j}$. For each player, it adjusts the probability assigned to each pure strategy in order to decrease the exploitability of the induced mixed strategy profile in the metagame.[5] This is called *exploitability descent (ED)* [77, p. 91, eq. 2.9; 65, 71]. This is repeated for K_{ED} sub-iterations. Second, it improves approximate BRs to the induced strategy profile in the full game. Specifically, it adjusts the pure strategies, especially those that happen to be BRs to the current approximate equilibrium (see the weighting scheme below), in order to increase the corresponding player's utility against that equilibrium. This is repeated for K_{BR} sub-iterations. MO maintains a set of strategies that is expanded on each iteration with BRs to the meta-strategies of the other players. These strategies are *static*, that is, they do not change after they are added. In contrast,

[5]Though exploitability involves a maximum and is thus not everywhere differentiable, it *is* continuous and almost-everywhere differentiable, so we use a subgradient.

our algorithm *dynamically* improves them over the course of training. Thus, the supports do not need to be augmented on each iteration, but instead improve autonomously over time. When the support sizes of the players are all 1, the metagame is trivial, and our algorithm reduces to simultaneous gradient dynamics.

Weighting. Instead of improving *only* the *best* response on each iteration, we can incentivize the other pure strategies to improve as well. To do this, we introduce the *rank-based mixing operator* mix.[6] It yields a weighted sum of its inputs: $\mathrm{mix}_{j\in J}\, x_j = \sum_{j\in J} w_j x_j$. The weight is $w_j = \frac{r_j}{|J|}$, where $r_j \in \{1, \ldots, |J|\}$ is the ordinal rank of x_j in increasing order (among equal elements, the one with the lower index gets the lower rank). This gives all elements some weight, but more weight to those that perform better. Thus it gives *all* elements some chance to improve, while simultaneously incentivizing specialization, that is, preventing uniformity. The intuition behind this is that an individual pure strategy does not need to perform well against all counter-strategies, but only against a subset thereof; it is *the ensemble as a whole* that must perform well against all of them. Since the best-performing pure strategies will be chosen automatically as responses, intuitively, a "good" ensemble will have pure strategies that are specialized. The weight of each element depends only on its rank, so the weights are invariant under strictly increasing transformations of the input values. The operator is shift-equivariant up to an additive term: $\mathrm{mix}(\mathbf{x} + \alpha\mathbf{1}) = \mathrm{mix}(\mathbf{x}) + \alpha \sum_{j\in J} w_j$. It is also scale-equivariant: $\mathrm{mix}(\alpha\mathbf{x}) = \alpha\,\mathrm{mix}(\mathbf{x})$ for $\alpha \geq 0$.

Pseudogradients. Our algorithm uses gradients of the utility functions. Some continuous-action games have discontinuous utility functions. To handle those cases, we can replace the corresponding gradient with a *pseudogradient*, which is the gradient of a smoothed version of the function (e.g., the function convolved with a narrow Gaussian). An unbiased estimator for this pseudogradient is obtained by evaluating the function at randomly-sampled perturbed points and using their values to approximate directional derivatives along those directions [30, 87, 106, 104, 9, 80]. For example, $\nabla_{\mathbf{x}}\, \mathrm{E}_{\mathbf{z}\sim D}\, f(\mathbf{x}+\sigma\mathbf{z}) = \mathrm{E}_{\mathbf{z}\sim D}\, \frac{1}{\sigma} f(\mathbf{x}+\sigma\mathbf{z})\mathbf{z}$ where D is the standard multivariate normal distribution with the same dimension as $\mathbf{x}$, so $\frac{1}{\sigma} f(\mathbf{x} + \sigma\mathbf{z})\mathbf{z}$ is an unbiased estimator of the gradient of the convolution of f with a Gaussian kernel of width σ. This approach is called Gaussian smoothing.

Utility smoothing. An alternative way to handle discontinuous utility functions is to use *smoothed surrogates*, that is, to directly smooth the utility functions themselves. We take this approach. For example, we replace "hard" argmaxes with softmaxes. The softmax function satisfies the property that $\mathrm{softmax}(\mathbf{x}/\tau) \to \mathrm{onehot}(\mathrm{argmax}(\mathbf{x}))$ as $\tau \to 0^+$, assuming no ties. Here, $\tau > 0$ can be interpreted as a temperature or *softness* parameter. There is also a probabilistic interpretation: $\mathrm{softmax}(\mathbf{x}) = \mathrm{E}_{\mathbf{z}\sim\mathrm{Gumbel}}\, \mathrm{onehot}(\mathrm{argmax}(\mathbf{x}+\mathbf{z}))$. Therefore, softmax is equivalent to the expected (one-hot) argmax of inputs perturbed with Gumbel noise.[7] There are software libraries that facilitate the construction of smooth surrogate proxies for objective functions. An example of such a library is SoftJAX [93], which we use. It implements various differentiable relaxations of selection, sorting, ranking, etc. from the literature [15, 16, 99, 121, 96].

Action parameterization. The games we test on have compact action spaces. We parameterize each type of action space as follows. We parameterize the unit n-cube using the logistic sigmoid function $\mathrm{logistic} : \mathbb{R}^n \to [0,1]^n$, defined by $\mathrm{logistic}(x)_i = \frac{1}{1+\exp(-x_i)}$. We parameterize the probability simplex $\triangle[n]$ using the softmax function $\mathrm{softmax} : \mathbb{R}^n \to \triangle[n]$.[8] We parameterize the unit n-ball using the squashing function $\mathrm{squash} : \mathbb{R}^n \to \mathbb{B}_n$, defined by $\mathrm{squash}(x) = \frac{x}{\sqrt{1+\|x\|^2}}$. These parameterizations allow actions to be arbitrarily close to the boundaries of the domain. This can be observed in some of the obtained strategy profiles, such as the one in Figure 15. We parameterize the unit circle in $\mathbb{R}^2$ by the map $x \mapsto x/\|x\|$ on nonzero vectors. The BR steps of Algorithm 2 update these parameters.

[6] This resembles the *ordered weighted averaging (OWA)* operator from the literature [122, 124, 123, 31, 26]. However, in our case, the weights need not sum to 1.

[7] A quantal response equilibrium (QRE) [78] is equivalent to a Bayesian Nash equilibrium (BNE) where players experience random, private perturbations to their payoffs [46, 41]. Similarly, Luce's choice axiom [66] is equivalent to a random utility model where a rational decision-maker's choices are influenced by Gumbel-distributed random perturbations [76].

[8] We also use this parameterization for the meta-strategy weights.

Table 1: Comparison to other methods in the literature: α-PSRO [84], JPSRO [68], NXDO [73], APSRO [75], SP-PSRO [74], MO [59], and RFP [36]. Methods that train approximate best responses (e.g., with reinforcement learning) count as not needing an exact BR oracle.

	α-PSRO	JPSRO	NXDO	APSRO	SP-PSRO	MO	RFP	**Ours**
n players	✓	✓	✗	✗	✗	✓	✓	✓
No exact BR oracle	✗	✗	✓	✓	✓	✗	✗	✓
No metagame oracle	✗	✗	✓	✓	✓	✗	✓	✓
NE as goal	✗	✗	✓	✓	✓	✓	✓	✓
Constant size	✗	✗	✗	✗	✗	✗	✗	✓

Size of the support sets. In our algorithm, the size of the pure strategy set for each player is a free hyperparameter that can be tuned according to the problem and the resources available to the solver. The same is true for other hyperparameters in machine learning and optimization, such as batch size, learning rate, hidden layer sizes, *etc*. Such hyperparameters are usually determined by trial and error. In our case, the size can be determined by the number of parallel workers available to the solver. This is because the utility function evaluations performed by Algorithm 2 in each sub-iteration can be done in parallel. If one has access to parallel computing infrastructure, one can scale *up* the sizes of the supports as much as possible, until all of this parallelism is used up, thus reaping the benefits of large supports while incurring little to no penalty in terms of wall time. Alternatively, it could be determined by known theoretical properties of the given game. For example, the game might satisfy some conditions for which there is a theoretical guarantee that it has an approximate mixed-strategy NE that mixes over just k pure strategies. Such guarantees exist in the literature, as described in §3.

In short, the size to choose in practice depends on multiple factors, including (1) the exploitability guarantee needed for a given application (for example, some games have no finite-support equilibria, but finite-support approximate equilibria for any positive exploitability), (2) theoretical guarantees from the literature about the support size needed to attain that exploitability, and the amount of computing resources in terms of (3) time and (4) memory available to the practitioner.

Comparison to prior work. Table 1 summarizes some of the main differences between our method and prior ones from the literature. As this table shows, our method is the first one, to our knowledge, that tackles the problem that we are interested in, in its full generality, making as few assumptions as possible (e.g., no BR or metagame equilibrium oracles). Other methods from the literature can tackle more specific settings, but are *not* able to tackle this problem in its full generality, and thus cannot be used as fair baselines in a like-for-like comparison. They have very different assumptions and domains of applicability.

5 Experiments

In this section, we use $K_{\text{ED}} = K_{\text{BR}} = 1$, and $\tau = 0.05$ for continuous Colonel Blotto and $\tau = 0.1$ for the auctions. Each experiment in this section runs for 2×10^7 iterations, with the exploitability evaluated once per epoch of 10^5 iterations, and we run 16 trials per experiment. In our plots, solid lines show the mean across trials, and bands show the standard error of the mean (SEM). In each legend, "lr" means the learning rate (i.e., the stepsize η) and "size" means the size of each player's support. The X axis shows the number of iterations of the algorithm (labeled "batch" in some plots). The Y axis shows the exploitability of the learned strategy profile *in the full game*.[9] Each experiment ran on one NVIDIA A100 SXM4 40GB GPU on a computer cluster running Springdale Linux 8.6. Again, we emphasize that it is not our goal to match or exceed the state of the art on a particular problem. Rather, our goal is to present a method that can tackle the problem *in its full generality*, with as few assumptions as possible. Nevertheless, we compare our method directly with MO and RFP (equipped with the oracles they require) on continuous Colonel Blotto in Appendix B.2, and we ablate

[9]The exploitability in the full game is approximated by performing, for each player's best response, a simple maximization of utility over a finite set of points in the player's action space: a fixed discretization of that space for the figures in this section, and, for the tables in Appendix B, 10^5 points obtained via the low-discrepancy sequences described in Pillards and Cools [97] and Roberts [102], together with the pure strategies in both players' supports (the game is symmetric, so either player can play any of them). Since this maximization is over finitely many points, it yields a lower bound on the true exploitability.

the use of exact NE and BR oracles within our method in Appendix B.3. Games with discontinuous utility functions are described in terms of their *smoothed* versions with softness parameter τ, which are used by the algorithm. Exploitability is also measured on these smoothed versions, except in Appendix B.

Interval game. Among the two-player games that have infinite sets of pure strategies available to the contestants, "games on the square" have received much attention [60, 44, 91, 92, 25, 112, 108]. These games generalize matrix games by replacing the elements M_{ij} of a utility matrix by a real-valued function $M(x, y)$ defined on $x, y \in [a, b]$ for some $a, b \in \mathbb{R}, a < b$. The pure strategies available to Players 1 and 2 are indexed by the real numbers x and y, while their mixed strategies become probability distributions over the interval. An example of such a game is the following. Let $I = \{1, 2\}$, $S_i = [0, 1]$, and $u_1(x, y) = -u_2(x, y) = (x - y)^2$. A version of this game on $[-1, 1]^2$ was studied by Parrilo [90, p. 2856]. Player 2 wants to be close to Player 1, so it tries to predict Player 1's number, while Player 1 tries to be unpredictable. The game has a unique NE, in which Player 1 picks from $\{0, 1\}$ with equal probability and Player 2 picks $\frac{1}{2}$. Player 1's expected utility is $\frac{1}{4}$. Results are shown in Figure 1. Our algorithm attains low exploitability only when Player 1 has at least 2 pure strategies to randomize over, which is expected since its equilibrium strategy mixes over 2 pure strategies. With a learning rate of 10^{-4}, every support size of at least 2 reaches an exploitability of about 2×10^{-3} to 3×10^{-3}.

Circle game. This resembles the interval game but on the unit circle, which is a homogeneous space. More precisely, let $I = \{1, 2\}$, $S_i \subset \mathbb{R}^2$ be the unit circle, and $u_1(x, y) = -u_2(x, y) = \|x - y\|_2^2$. Each player picking uniformly on the unit circle is an NE. Results are shown in Figure 2. For each learning rate, larger support sizes monotonically yield lower exploitability.

Glicksberg–Gross game. This is a game on the square that was introduced and analyzed by Glicksberg and Gross [40]. Let $I = \{1, 2\}$, $S_i = [0, 1]$, and $u_1(x, y) = -u_2(x, y) = \frac{(1+x)(1+y)(1-xy)}{(1+xy)^2}$. It has a unique mixed-strategy NE, in which the strategy of each player has cumulative distribution function $F(t) = \frac{4}{\pi} \arctan \sqrt{t}$ for $t \in [0, 1]$, and Player 1's expected utility is $\frac{4}{\pi}$. The authors use this example to illustrate that, unlike games with polynomial utility functions, games with rational utility functions do not always have equilibria that are concentrated on finitely many points. Results are shown in Figure 3. For each learning rate, larger support sizes yield lower exploitability.

Continuous Colonel Blotto. A Colonel Blotto game is a game in which players distribute limited resources over multiple battlefields (i.e., items). A battlefield is won by whoever devotes the most resources to it. A player's utility is the number of battlefields it wins. It models real-world situations of conflict or competition that involve *resource allocation*, such as political campaigns, research and development, national security, and systems defense. The original Colonel Blotto game was introduced and studied by Borel [18, 19]. Subsequently, many variants have been introduced and studied in the literature. Gross and Wagner [45] obtained exact solutions for various cases of this continuous game, including all 2-battlefield cases (with possibly unequal budgets) and all 3-battlefield cases with equal budgets. Boix-Adserà et al. [17] analyzed a multiplayer generalization of this. Formally, our game's utility function is $u : (\triangle[k])^n \to \mathbb{R}^n$ where n is the number of players, k is the number of battlefields, $u(\mathbf{A}) = \text{player-softmax}(\mathbf{A}/\tau) \cdot \mathbf{1}_k$, and player-softmax applies the softmax across players, separately for each battlefield. Results are shown in Figures 4–5. Larger support sizes yield lower exploitability.

Security game. In game theory, a coordination game is one that rewards players choosing the same action. In contrast, an *anti*-coordination game *penalizes* choosing the same action. A *dis*coordination game is a hybrid of the two. Specifically, one player is incentivized to choose the same action as the other player, while the other player is incentivized to choose a different action. Such games, in general, lack pure-strategy NE. The canonical example of such a game is the 2×2 *matching pennies* game, whose utility matrix is (a positive affine transformation of) the identity matrix. To extend this game to a continuous action space, we replace the notion of choosing the *same* action with that of choosing a *nearby* action, under some metric and distance scale. More precisely, let $I = \{1, 2\}$, $S_i = [0, 1]^d$, where $d \in \{1, 2\}$ is the dimension, and $u_1(x, y) = -u_2(x, y) = \exp(-\sigma^{-2}\|x - y\|_2^2)$, where $\sigma > 0$ is a distance scale (we use $\sigma = 1$). This game, which is described in Washburn [119, p. 85] as a *search game*, models a situation where Player 1 tries to detect or capture Player 2, and its

probability of doing so decays with distance. It resembles a security game on a continuous space. Such continuous security games have been studied in prior work [53–55]. Our results for this game are shown in Figures 6 and 7. Larger support sizes generally yield lower exploitability, except for a learning rate of 10^{-4} and size of 2, which yields the lowest exploitability for the 1D security game. This might be because, with $\sigma = 1$, the 1D game has an exact equilibrium in which Player 1 plays $\frac{1}{2}$ and Player 2 plays 0 and 1 with equal probability, so 2 pure strategies per player suffice.

All-pay auction. An auction is a mechanism by which one or more items are sold to one or more bidders. Auctions play a central role in the study of markets and are used in a wide range of contexts. In a single-item sealed-bid auction, bidders simultaneously submit bids and the highest bidder wins the item. In an *all-pay* auction, each player always pays its bid. Such auctions are widely used to model contests, including rent seeking through lobbying in regulated or trade-protected industries, technology and R&D races, political campaigns, and job promotions [8]. In a *complete-information* all-pay auction, the valuations of the players for all items are common knowledge. Such an auction lacks pure-strategy equilibria [8]. We consider the case where every player values the item at 1. In the two-player case, in equilibrium, the players bid uniformly on $[0, 1]$. Our game's utility function is $u : [0, 1]^n \to \mathbb{R}^n$ where n is the number of players and $u(\mathbf{a}) = \text{softmax}(\mathbf{a}/\tau) - \mathbf{a}$, which is a smooth approximation to the original game. Results are shown in Figures 8–10. Larger support sizes generally yield lower exploitability.

Chopstick auction. Multi-item auctions are of great importance in practice, for example in strategic sourcing [105] and radio spectrum allocation [81, 82]. However, deriving equilibrium bidding strategies for multi-item auctions is notoriously difficult. A rare notable instance where equilibrium strategies have been derived is the *chopstick auction* [111, 110]. In this auction, 3 chopsticks are sold simultaneously in separate first-price sealed-bid auctions. There are 2 bidders, and it is common knowledge that a pair of chopsticks is worth \$1, a single chopstick is worth nothing by itself, and 3 chopsticks are worth the same as 2. Here, pure strategies are triples of nonnegative real numbers (bids). This game has an interesting equilibrium [110, 111]. Let T be the regular tetrahedron that is the convex hull of the four points $(\frac{1}{2}, \frac{1}{2}, 0)$, $(\frac{1}{2}, 0, \frac{1}{2})$, $(0, \frac{1}{2}, \frac{1}{2})$, and $(0, 0, 0)$. The uniform distribution on the surface of T generates a symmetric equilibrium. Furthermore, all points inside the tetrahedron are pure BRs to this equilibrium mixture. We benchmark on the chopstick auction since it is a rare case of a multi-item auction with a known analytic equilibrium (shown in Figure 17). It is also a canonical case of simultaneous separate auctions under combinatorial preferences. Formally, our game's utility function is $u : ([0, 1]^3)^2 \to \mathbb{R}^2$ where $u(\mathbf{A}) = \text{logistic}((\mathbf{W} \cdot \mathbf{1}_3 - 1.5)/\tau) - (\mathbf{A} \odot \mathbf{W}) \cdot \mathbf{1}_3$ and $\mathbf{W} = \text{player-softmax}(\mathbf{A}/\tau)$. Results are shown in Figure 11. Larger support sizes generally yield lower exploitability.

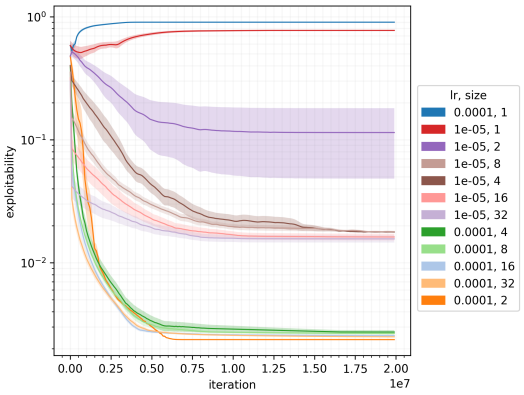


Figure 1: Interval game.

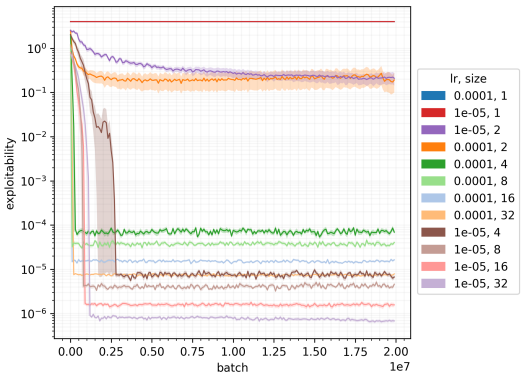


Figure 2: Circle game.

6 Conclusions and Future Research

We introduced an algorithm for computing approximate mixed-strategy NE of multiplayer continuous-action games. It gradually improves a metagame equilibrium while simultaneously and gradually

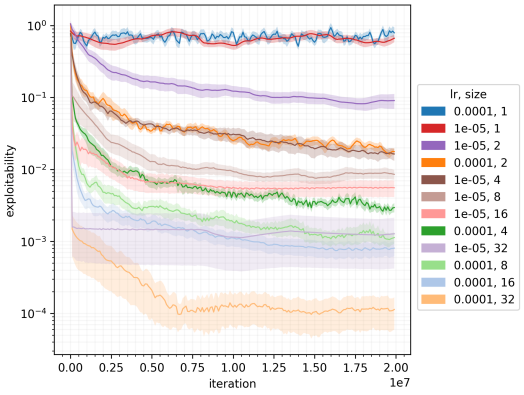


Figure 3: Glicksberg–Gross game.

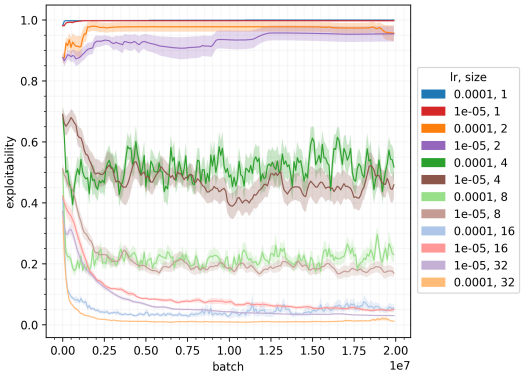


Figure 4: 2-player, 3-item Colonel Blotto.

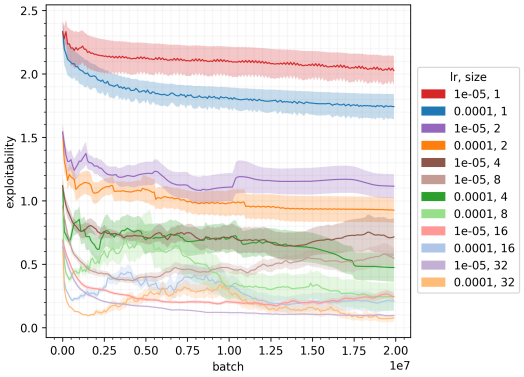


Figure 5: 3-player, 3-item Colonel Blotto.

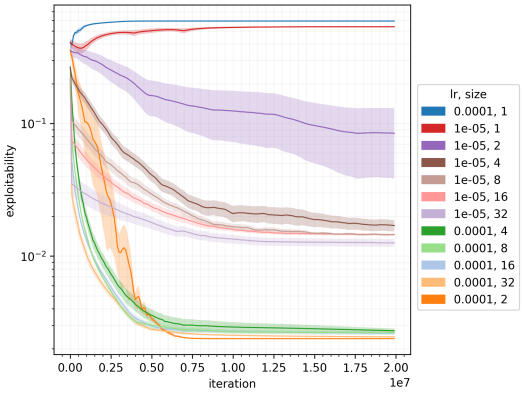


Figure 6: 1D security game.

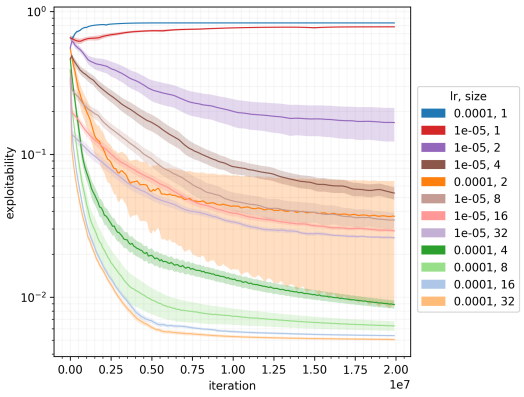


Figure 7: 2D security game.

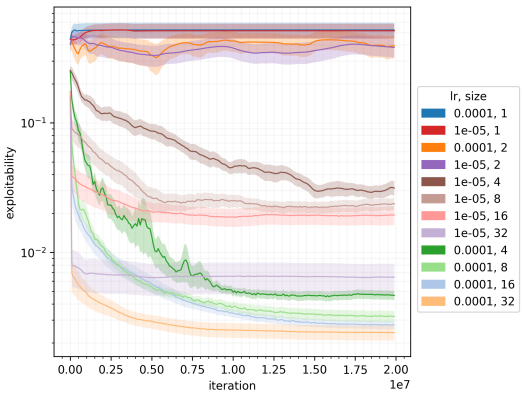


Figure 8: 2-player all-pay auction.

improving pure strategies against it. It mitigates the need for exact equilibrium and/or global best-response computation on each iteration. Furthermore, it requires only constant memory in the number of iterations. We tested our algorithm on various continuous-action games. It obtained approximate mixed-strategy NE with low exploitability. In some games, the quality of the approximation depended, unsurprisingly, on the number of pure strategies available to each player to mix over. The exploitability plots showed that our algorithm converges to low exploitability when the sizes of the players' supports are sufficiently large, with larger supports allowing for a better approximation to an NE for the underlying game. As expected, a very small support size (e.g., having a single pure strategy in each player's strategy set) may cause a failure to converge to low exploitability. With a limited number

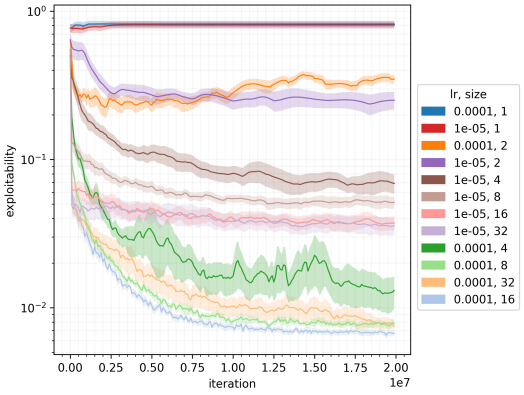


Figure 9: 3-player all-pay auction.

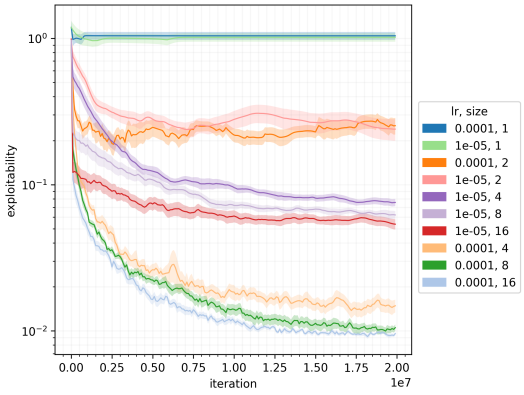


Figure 10: 4-player all-pay auction.

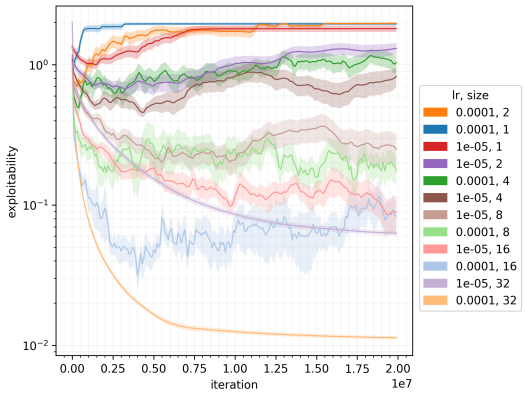


Figure 11: Chopstick auction.

of pure strategies, it might not be possible to even *represent* an exact equilibrium of the game. For example, the Glicksberg–Gross game has no finite-support equilibrium. Our work also has limitations. We do not provide a general convergence guarantee for our method (Appendix A contains a partial theoretical analysis), and the support sizes must currently be chosen in advance. Also, for games with discontinuous utilities, the exploitability in §5 is that of the smoothed games.

A potential direction for future research would be to explore dynamically growing the strategy sets over the course of training, but in a controlled way (as opposed to on every iteration, like MO). For example, when we want to grow a strategy set, we can duplicate one of its elements and continue training.[10] Alternatively, one can double the size of the strategy set by duplicating each strategy and continuing training. It then remains to decide *when* to grow the strategy sets. For example, this could be done when the estimated exploitability of the learned strategy profile in the full game has stopped decreasing for some period of time. This is similar to learning rate scheduling, in which, for example, the learning rate is reduced when a monitored metric plateaus [98, 107, 85, 3].

Acknowledgments and Disclosure of Funding

This material is based on work supported by the Vannevar Bush Faculty Fellowship ONR N00014-23-1-2876, National Science Foundation grants RI-2312342 and RI-1901403, ARO award W911NF2210266, and NIH award A240108S001.

[10]The fact that the weights assigned by the mix operator described in §4 are based on ordinal ranks (which differ even between equal elements) ensures symmetry breaking—that is, two pure strategies can diverge even if they are initially equal.

References

[1] Lukáš Adam, Rostislav Horčík, Tomáš Kasl, and Tomáš Kroupa. Double oracle algorithm for computing equilibria in continuous games. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

[2] Salam Afiouni, Jakub Černý, Chun Kai Ling, and Christian Kroer. Commitment to sparse strategies in two-player games. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2025.

[3] Ayman Al-Kababji, Faycal Bensaali, and Sarada Prasad Dakua. Scheduling techniques for liver segmentation: ReduceLROnPlateau vs OneCycleLR. In *International Conference on Intelligent Systems and Pattern Recognition*, 2022.

[4] Ingo Althöfer. On sparse approximations to randomized strategies and convex combinations. *Linear Algebra and its Applications*, 1994.

[5] Martin S. Andersen, Joachim Dahl, and Lieven Vandenberghe. CVXOPT: Python software for convex optimization, 2026. URL `https://cvxopt.org`. Version 1.3.3.

[6] Robert Aumann. Subjectivity and correlation in randomized strategies. *Journal of Mathematical Economics*, 1:67–96, 1974.

[7] Yakov Babichenko, Siddharth Barman, and Ron Peretz. Empirical distribution of equilibrium play and its testing application. *Mathematics of Operations Research*, 2017.

[8] Michael Baye, Dan Kovenock, and Casper de Vries. The all-pay auction with complete information. *Economic Theory*, 1996.

[9] Albert Berahas, Liyuan Cao, Krzysztof Choromanski, and Katya Scheinberg. A theoretical and empirical comparison of gradient approximations in derivative-free optimization. *Foundations of Computational Mathematics*, 2022.

[10] Ulrich Berger. Brown's original fictitious play. *Journal of Economic Theory*, 135(1):572–578, 2007.

[11] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. arXiv:1912.06680, 2019.

[12] Martin Bichler, Maximilian Fichtl, Stefan Heidekrüger, Nils Kohring, and Paul Sutterer. Learning equilibria in symmetric auction games using artificial neural networks. *Nature Machine Intelligence*, 3(8):687–695, 2021.

[13] Martin Bichler, Nils Kohring, and Stefan Heidekrüger. Learning equilibria in asymmetric auction games. *INFORMS Journal on Computing*, 35(3):523–542, 2023.

[14] Martin Bichler, Max Fichtl, and Matthias Oberlechner. Computing Bayes–Nash equilibrium strategies in auction games via simultaneous online dual averaging. *Operations Research*, 73 (2):1102–1127, 2025.

[15] Mathieu Blondel, Vivien Seguy, and Antoine Rolet. Smooth and sparse optimal transport. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 880–889, 2018.

[16] Mathieu Blondel, Olivier Teboul, Quentin Berthet, and Josip Djolonga. Fast differentiable sorting and ranking. In *International Conference on Machine Learning (ICML)*, 2020.

[17] Enric Boix-Adserà, Benjamin Edelman, and Siddhartha Jayanti. The multiplayer Colonel Blotto game. *Games and Economic Behavior*, 2021.

[18] Émile Borel. La théorie du jeu et les équations intégrales à noyau symétrique gauche. *Comptes rendus de l'Académie des Sciences*, 173:1304–1308, 1921.

[19] Émile Borel. The theory of play and integral equations with skew symmetric kernels. *Econometrica*, 1953.

[20] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: Composable transformations of Python+NumPy programs, 2018. URL `http://github.com/jax-ml/jax`.

[21] George W. Brown. Iterative solution of games by fictitious play. In Tjalling C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 374–376. John Wiley & Sons, 1951.

[22] Noam Brown and Tuomas Sandholm. Regret transfer and parameter optimization. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 28, pages 594–601. AAAI Press, 2014.

[23] Noam Brown and Tuomas Sandholm. Simultaneous abstraction and equilibrium finding in games. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 489–496, Palo Alto, CA, 2015. AAAI Press.

[24] Bill Chen and Jerrod Ankenman. *The Mathematics of Poker*. ConJelCo, Pittsburgh, PA, 2006.

[25] H. Chin, Thiruvenkatachari Parthasarathy, and Thirukkannamangai Raghavan. Optimal strategy sets for continuous two person games. *Sankhyā: The Indian Journal of Statistics*, 1976.

[26] Orsolya Csiszár. Ordered weighted averaging operators: a short review. *IEEE Systems, Man, and Cybernetics Magazine*, 7(2):4–12, 2021.

[27] Partha Dasgupta and Eric Maskin. The existence of equilibrium in discontinuous economic games, I: Theory. *Review of Economic Studies*, 53(1):1–26, 1986.

[28] Gérard Debreu. A social equilibrium existence theorem. *Proceedings of the National Academy of Sciences*, 38(10):886–893, 1952.

[29] DeepMind, Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Laurent Sartran, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Miloš Stanojević, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola. The DeepMind JAX Ecosystem, 2020. URL `http://github.com/google-deepmind`.

[30] John Duchi, Michael Jordan, Martin Wainwright, and Andre Wibisono. Optimal rates for zero-order convex optimization: The power of two function evaluations. *IEEE Transactions on Information Theory*, 2015.

[31] Ali Emrouznejad and Marianna Marra. Ordered weighted averaging operators 1988–2014: A citation-based literature survey. *International Journal of Intelligent Systems*, 29(11):994–1014, 2014.

[32] Ky Fan. Fixed point and minimax theorems in locally convex topological linear spaces. *Proceedings of the National Academy of Sciences*, 38(2):121–126, 1952.

[33] Sevan Ficici and Jordan Pollack. A game-theoretic memory mechanism for coevolution. In *Genetic and Evolutionary Computation Conference (GECCO)*, pages 286–297. Springer Berlin Heidelberg, 2003.

[34] Sevan Gregory Ficici. *Solution concepts in coevolutionary algorithms*. PhD thesis, Brandeis University, 2004.

[35] Sjur Didrik Flåm and Andrzej Ruszczyński. Noncooperative convex games: computing equilibrium by partial regularization. Technical report, International Institute for Applied Systems Analysis, 1994.

[36] Sam Ganzfried. Algorithm for computing approximate Nash equilibrium in continuous games with application to continuous Blotto. *Games*, 2021.

[37] Sam Ganzfried and Tuomas Sandholm. Computing equilibria by incorporating qualitative models. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2010.

[38] Papiya Ghosh and Rajendra P. Kundu. Best-shot network games with continuous action space. *Research in Economics*, 2019.

[39] Irving Glicksberg. A further generalization of the Kakutani fixed point theorem, with application to Nash equilibrium points. *Proceedings of the American Mathematical Society*, 1952.

[40] Irving Glicksberg and Oliver Gross. Notes on games over the square. In Harold William Kuhn and Albert William Tucker, editors, *Contributions to the Theory of Games, Volume II*, pages 173–182. Princeton University Press, 1953.

[41] Jacob K. Goeree, Charles A. Holt, and Thomas R. Palfrey. *Quantal Response Equilibrium: A Stochastic Theory of Games*. Princeton University Press, 2016.

[42] Denizalp Goktas and Amy Greenwald. Exploitability minimization in games and beyond. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[43] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[44] Oliver Alfred Gross. A rational game on the square. In Melvin Dresher, Albert William Tucker, and Philip Wolfe, editors, *Contributions to the Theory of Games, Volume III*, pages 307–312. Princeton University Press, 1957.

[45] Oliver Alfred Gross and R. A. Wagner. *A continuous Colonel Blotto game*. RAND Corporation, 1950.

[46] John C. Harsanyi. Games with randomly disturbed payoffs: A new rationale for mixed-strategy equilibrium points. *International Journal of Game Theory*, 2(1):1–23, 1973.

[47] Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2023. URL `http://github.com/google/flax`.

[48] Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. arXiv:1603.01121, 2016.

[49] Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *International Conference on Machine Learning (ICML)*, pages 805–813. JMLR, 2015.

[50] Sébastien Hémon, Michel de Rougemont, and Miklos Santha. Approximate Nash equilibria for multi-player games. In *International Symposium on Algorithmic Game Theory (SAGT)*, 2008.

[51] Q. Huangfu and J. A. J. Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018.

[52] John D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[53] Nitin Kamra, Fei Fang, Debarun Kar, Yan Liu, and Milind Tambe. Handling continuous space security games with neural networks. In *First International Workshop on Artificial Intelligence in Security (IWAISe)*, 2017.

[54] Nitin Kamra, Umang Gupta, Fei Fang, Yan Liu, and Milind Tambe. Policy learning for continuous space security games using neural networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[55] Nitin Kamra, Umang Gupta, Kai Wang, Fei Fang, Yan Liu, and Milind Tambe. DeepFP for finding Nash equilibrium in continuous action spaces. In *International Conference on Decision and Game Theory for Security (GameSec)*, 2019.

[56] Christian Kroer and Tuomas Sandholm. Extensive-form game abstraction with bounds. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, pages 621–638, New York, NY, USA, 2014. Association for Computing Machinery.

[57] Christian Kroer and Tuomas Sandholm. Discretization of continuous action spaces in extensive-form games. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2015.

[58] Christian Kroer and Tuomas Sandholm. Imperfect-recall abstractions with bounds in games. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, pages 459–476, New York, NY, USA, 2016. Association for Computing Machinery.

[59] Tomáš Kroupa and Tomáš Votroubek. Multiple oracle algorithm to solve continuous games. In *International Conference on Decision and Game Theory for Security (GameSec)*, 2022.

[60] Harold William Kuhn and Albert William Tucker, editors. *Contributions to the Theory of Games, Volume II*. Princeton University Press, 1953.

[61] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[62] David S. Leslie and Edmund J. Collins. Generalised weakened fictitious play. *Games and Economic Behavior*, 56(2):285–298, 2006.

[63] Zun Li and Michael Wellman. Evolution strategies for approximate solution of Bayesian games. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 5531–5540, 2021.

[64] Richard J. Lipton, Evangelos Markakis, and Aranyak Mehta. Playing large games using simple strategies. In *Proceedings of the 4th ACM Conference on Electronic Commerce*, 2003.

[65] Edward Lockhart, Marc Lanctot, Julien Pérolat, Jean-Baptiste Lespiau, Dustin Morrill, Finbarr Timbers, and Karl Tuyls. Computing approximate equilibria in sequential adversarial games by exploitability descent. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

[66] R. Duncan Luce. *Individual Choice Behavior: A Theoretical Analysis*. John Wiley & Sons, 1959.

[67] Alberto Marchesi, Francesco Trovò, and Nicola Gatti. Learning probably approximately correct maximin strategies in simulation-based games with infinite strategy spaces. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2020.

[68] Luke Marris, Paul Muller, Marc Lanctot, Karl Tuyls, and Thore Graepel. Multi-agent training beyond zero-sum with correlated equilibrium meta-solvers. In *International Conference on Machine Learning (ICML)*, 2021.

[69] Carlos Martin and Tuomas Sandholm. Finding mixed-strategy equilibria of continuous-action games without gradients using randomized policy networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2844–2852. International Joint Conferences on Artificial Intelligence Organization, 2023.

[70] Carlos Martin and Tuomas Sandholm. ApproxED: Approximate exploitability descent via learned best responses. arXiv:2301.08830v3, 2024.

[71] Carlos Martin and Tuomas Sandholm. ApproxED: Approximate exploitability descent via learned best responses. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1454–1463, 2025.

[72] Stephen McAleer, John B. Lanier, Roy Fox, and Pierre Baldi. Pipeline PSRO: A scalable approach for finding approximate Nash equilibria in large games. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 20238–20248, 2020.

[73] Stephen McAleer, John B. Lanier, Kevin A. Wang, Pierre Baldi, and Roy Fox. XDO: A double oracle algorithm for extensive-form games. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[74] Stephen McAleer, John B. Lanier, Kevin Wang, Pierre Baldi, Roy Fox, and Tuomas Sandholm. Self-play PSRO: Toward optimal populations in two-player zero-sum games. arXiv:2207.06541, 2022.

[75] Stephen McAleer, Kevin Wang, John Lanier, Marc Lanctot, Pierre Baldi, Tuomas Sandholm, and Roy Fox. Anytime PSRO for two-player zero-sum games. arXiv:2201.07700, 2022.

[76] Daniel McFadden. Conditional logit analysis of qualitative choice behavior. In Paul Zarembka, editor, *Frontiers in Econometrics*, pages 105–142. Academic Press, 1973.

[77] Richard D. McKelvey and Andrew McLennan. Computation of equilibria in finite games. In Hans M. Amman, David A. Kendrick, and John Rust, editors, *Handbook of Computational Economics*, volume 1, chapter 2, pages 87–142. Elsevier, 1996.

[78] Richard D. McKelvey and Thomas R. Palfrey. Quantal response equilibria for normal form games. *Games and Economic Behavior*, 10(1):6–38, 1995.

[79] H. Brendan McMahan, Geoffrey J. Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *International Conference on Machine Learning (ICML)*, 2003.

[80] Luke Metz, C. Daniel Freeman, Samuel S. Schoenholz, and Tal Kachman. Gradients are not all you need. arXiv:2111.05803, 2021.

[81] Paul Milgrom and Ilya Segal. Deferred-acceptance auctions and radio spectrum reallocation. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, 2014.

[82] Paul Milgrom and Ilya Segal. Clock auctions and radio spectrum reallocation. *Journal of Political Economy*, 2020.

[83] Paul Milgrom and Robert Weber. Distributional strategies for games with incomplete information. *Mathematics of Operations Research*, 10:619–632, 1985.

[84] Paul Muller, Shayegan Omidshafiei, Mark Rowland, Karl Tuyls, Julien Perolat, Siqi Liu, Daniel Hennes, Luke Marris, Marc Lanctot, Edward Hughes, Zhe Wang, Guy Lever, Nicolas Heess, Thore Graepel, and Remi Munos. A generalized training approach for multiagent learning. In *International Conference on Learning Representations (ICLR)*, 2020.

[85] Kensuke Nakamura, Bilel Derbel, Kyoung-Jae Won, and Byung-Woo Hong. Learning-rate annealing methods for deep neural networks. *Electronics*, 2021.

[86] John Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950.

[87] Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 2017.

[88] Frans Oliehoek, Edwin De Jong, and Nikos Vlassis. The parallel Nash memory for asymmetric games. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pages 337–344. Association for Computing Machinery, 2006.

[89] Shayegan Omidshafiei, Christos Papadimitriou, Georgios Piliouras, Karl Tuyls, Mark Rowland, Jean-Baptiste Lespiau, Wojciech Czarnecki, Marc Lanctot, Julien Perolat, and Remi Munos. α-Rank: Multi-agent evaluation by evolution. *Scientific Reports*, 9(1):9937, 2019.

[90] Pablo Parrilo. Polynomial games and sum of squares optimization. In *IEEE Conference on Decision and Control (CDC)*, 2006.

[91] Thiruvenkatachari Parthasarathy. On games over the unit square. *SIAM Journal on Applied Mathematics*, 1970.

[92] Thiruvenkatachari Parthasarathy and Thirukkannamangai Raghavan. Equilibria of continuous two-person games. *Pacific Journal of Mathematics*, 1975.

[93] Anselm Paulus, A. René Geist, Vít Musil, Sebastian Hoffmann, Onur Beker, and Georg Martius. SoftJAX & SoftTorch: Empowering automatic differentiation libraries with informative gradients. *arXiv preprint arXiv:2603.08824*, 2026.

[94] Steven Perkins and David S. Leslie. Stochastic fictitious play with continuous action sets. *Journal of Economic Theory*, 152:179–213, 2014.

[95] Steven Perkins, Panayotis Mertikopoulos, and David S. Leslie. Mixed-strategy learning with continuous action sets. *IEEE Transactions on Automatic Control*, 62(1):379–384, 2017.

[96] Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. Differentiable sorting networks for scalable sorting and ranking supervision. In *International Conference on Machine Learning (ICML)*, pages 8546–8555, 2021.

[97] Tim Pillards and Ronald Cools. Transforming low-discrepancy sequences from a cube to a simplex. *Journal of Computational and Applied Mathematics*, 2005.

[98] Samira Pouyanfar and Shu-Ching Chen. T-LRA: Trend-based learning rate annealing for deep neural networks. In *2017 IEEE Third International Conference on Multimedia Big Data (BigMM)*, 2017.

[99] Sebastian Prillo and Julian Martin Eisenschlos. SoftSort: A continuous relaxation for the argsort operator. In *International Conference on Machine Learning (ICML)*, 2020.

[100] John Rehbeck. Note on unique Nash equilibrium in continuous games. *Games and Economic Behavior*, 110:216–225, 2018.

[101] Brian Roberson. The Colonel Blotto game. *Economic Theory*, 2006.

[102] Martin Roberts. The unreasonable effectiveness of quasirandom sequences, 2018. URL `https://extremelearning.com.au/unreasonable-effectiveness-of-quasirandom-sequences/`.

[103] J. Ben Rosen. Existence and uniqueness of equilibrium points for concave n-person games. *Econometrica*, 33(3):520–534, 1965.

[104] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. arXiv:1703.03864, 2017.

[105] Tuomas Sandholm. Very-large-scale generalized combinatorial multi-attribute auctions: Lessons from conducting $60 billion of sourcing. In Nir Vulkan, Alvin E. Roth, and Zvika Neeman, editors, *The Handbook of Market Design*, pages 379–412. Oxford University Press, Oxford, United Kingdom, 2013.

[106] Ohad Shamir. An optimal algorithm for bandit and zero-order convex optimization with two-point feedback. *Journal of Machine Learning Research*, 2017.

[107] Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

[108] Noah Stein, Asuman Ozdaglar, and Pablo Parrilo. Separable and low-rank continuous games. *International Journal of Game Theory*, 37:475–504, 2008.

[109] Eric Steinberger, Adam Lerer, and Noam Brown. DREAM: Deep regret minimization with advantage baselines and model-free learning. arXiv:2006.10410, 2020.

[110] Balázs Szentes and Robert W. Rosenthal. Beyond chopsticks: Symmetric equilibria in majority auction games. *Games and Economic Behavior*, 2003.

[111] Balázs Szentes and Robert W. Rosenthal. Three-object two-bidder simultaneous auctions: chopsticks and tetrahedra. *Games and Economic Behavior*, 2003.

[112] Jenő Szép and Ferenc Forgó. Games played over the unit square. In *Introduction to the Theory of Games*, pages 196–199. Springer Netherlands, 1985.

[113] Finbarr Timbers, Nolan Bard, Edward Lockhart, Marc Lanctot, Martin Schmid, Neil Burch, Julian Schrittwieser, Thomas Hubert, and Michael Bowling. Approximate exploitability: Learning a best response. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2022.

[114] Guido Van Rossum and Fred L. Drake, Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[115] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 2019.

[116] Pauli Virtanen et al. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272, 2020.

[117] Michael Walton and Viliam Lisý. Multi-agent reinforcement learning in OpenSpiel: A reproduction report. arXiv:2103.00187, 2021.

[118] Guillaume Wang and Lénaïc Chizat. An exponentially converging particle method for the mixed Nash equilibrium of continuous games. *Open Journal of Mathematical Optimization*, 6: 1–66, 2025.

[119] Alan Washburn. Games with a continuum of strategies. In *Two-Person Zero-Sum Games*, pages 83–106. Springer US, 2014.

[120] Daan Wierstra, Tom Schaul, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. In *IEEE Congress on Evolutionary Computation*, pages 3381–3387. IEEE, 2008.

[121] Yujia Xie, Hanjun Dai, Minshuo Chen, Bo Dai, Tuo Zhao, Hongyuan Zha, Wei Wei, and Tomas Pfister. Differentiable top-k with optimal transport. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[122] Ronald R. Yager. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):183–190, 1988.

[123] Ronald R. Yager and Janusz Kacprzyk, editors. *The Ordered Weighted Averaging Operators: Theory and Applications*. Springer, 1997.

[124] Ronald R. Yager, Janusz Kacprzyk, and Gleb Beliakov, editors. *Recent Developments in the Ordered Weighted Averaging Operators: Theory and Practice*. Springer, 2011.

A Theoretical Analysis

In this section, we present a theoretical analysis of our approach. A general convergence proof is beyond the scope of this paper, though whether our method converges in general is a potentially interesting question for future research.

A.1 Exploitability Decomposition

In this subsection, we show that the exploitability of the strategy profile in the full game can be decomposed into two nonnegative components, which our method reduces with separate steps.

Let F be the full game. Let M be the subgame induced by the current supports, that is, the metagame. We identify each of these games with its mixed extension, so that F_i and M_i denote player i's sets of mixed strategies in the full game and in the subgame, respectively, and u_i denotes player i's expected utility ($\bar{u}_i$ in the notation of §2). Let s be the current strategy profile. The current strategy profile is supported on the subgame. That is, $s_i \in M_i$ for all $i \in I$. Furthermore, each strategy set of the subgame is a subset of the corresponding strategy set of the full game. That is, $M_i \subseteq F_i$ for all $i \in I$.

The exploitability of the strategy profile in the full game, $E_F(s)$, can be decomposed into two components: 1) the exploitability of the strategy profile in the subgame, $E_M(s)$—that is, the metagame exploitability—and 2) the *BR gap* between the full game and the subgame:

$$
\begin{aligned}
E_F(s) &= \sum_{i \in I} \left(\sup_{x \in F_i} u_i(s[i \mapsto x]) - u_i(s) \right) && (1) \\
&= \sum_{i \in I} \left(\sup_{x \in F_i} u_i(s[i \mapsto x]) - \sup_{x \in M_i} u_i(s[i \mapsto x]) \right. \\
&\qquad \left. + \sup_{x \in M_i} u_i(s[i \mapsto x]) - u_i(s) \right) && (2) \\
&= \sum_{i \in I} \left(\sup_{x \in F_i} u_i(s[i \mapsto x]) - \sup_{x \in M_i} u_i(s[i \mapsto x]) \right) \\
&\qquad + \sum_{i \in I} \left(\sup_{x \in M_i} u_i(s[i \mapsto x]) - u_i(s) \right) && (3) \\
&= \text{BR gap} + E_M(s). && (4)
\end{aligned}
$$

The BR gap measures how well best responses perform when constrained to lie within the existing supports, versus when allowed to vary over the full game. Both components are nonnegative: the BR gap because $M_i \subseteq F_i$, and $E_M(s)$ because $s_i \in M_i$. Therefore, the exploitability in the full game is low if and only if both components are low. That is, to guarantee low exploitability in the full game, it is necessary and sufficient to keep the subgame exploitability small, as well as to keep supported best responses close to true optimality in the full game. This motivates the algorithm presented in this paper, which simultaneously decreases the subgame exploitability by adjusting weights *and* moves the supports themselves so that they are closer to being "true best responses" in the full game.

A.2 Exploitability Minimization for the Metagame

In this subsection, we describe theoretical guarantees for exploitability descent in the metagame. One component of our algorithm is gradually performing exploitability descent on the metagame on each iteration. Assuming the supports are fixed, we can analyze the conditions under which exploitability descent yields an equilibrium for the metagame.

In the literature, there are many strong theoretical guarantees that hold for optimization under the assumption of convexity. Therefore, if the exploitability function is convex in the variables being optimized, those guarantees apply. Martin and Sandholm [71] prove the convexity of the exploitability function for a wide class of games, and also describe conditions under which approximate subgradient descent attains a global minimum of the exploitability function—that is, a Nash equilibrium, if one exists (see the theoretical analysis in Appendix B of the arXiv version [70]). For example, any constant-sum polymatrix game has convex exploitability, as does any two-player constant-sum

concave-convex game. These results apply to the metagame step of our method only under their conditions: the metagame must belong to such a class (as it does whenever the full game is two-player and constant-sum, but not, e.g., for three-player Colonel Blotto), the supports must be fixed, and the weights must be updated directly on the probability simplex, with suitable stepsizes. With the softmax parameterization we use, exploitability is generally not convex in the logits, so the guarantees do not carry over directly.

Some prior works related to exploitability descent are as follows. Lockhart et al. [65] analyzed exploitability descent in finite two-player zero-sum games in extensive form. As Goktas and Greenwald [42] note, minimizing exploitability is a natural way to compute NE, particularly when exploitability is convex, as it is in the pseudo-games obtained from a monotone pseudo-game by replacing each player's utility function with its second-order Taylor approximation [35].

A.3 BR Gap Minimization

To reduce the BR gap, we adjust the supports in the full game using gradient-based optimization to improve the performance of the best responses to the current strategy profile. Standard results for gradient-based optimization apply to this step only while the other players' strategies are held fixed, and, since utilities need not be concave, they guarantee only convergence to stationary points, so the BR gap need not vanish. If the game's utility function is discontinuous, we can use pseudogradients. Theoretical guarantees for optimization with such pseudogradients can be found in Nesterov and Spokoiny [87], but only for functions that are Lipschitz continuous or have Lipschitz continuous gradients, which excludes discontinuous utilities.

B Additional Tables and Figures

In this section, we include additional tables and figures that did not fit in the body of the paper.

The tables are all for continuous Colonel Blotto. Their results for our method come from a later implementation of it, which differs from Algorithm 2 in two ways. First, in its ED and BR steps, it updates the logits of each meta-strategy and pure strategy using the gradient with respect to that strategy itself, rather than with respect to its logits; since both kinds of strategies lie in simplices parameterized by softmax, each step is an exponentiated-gradient (multiplicative-weights) update. Second, it constructs the metagame from the original (unsmoothed) game, and uses the smoothed game, with $\tau = 0.01$, only for the gradients of its BR steps. Unlike in §5, exploitability is measured on the original game. Each run has 16 trials, which run in parallel on one GPU of the type used in §5, and stops after 10 minutes of training. In each table, Mean, SEM, and Std summarize the final exploitability across the 16 trials: Std is the population standard deviation, and SEM is the standard error of the mean computed from the sample standard deviation (i.e., with Bessel's correction), so $\text{SEM} = \text{Std}/\sqrt{15}$.

B.1 Higher Action Dimensionalities and Support Sizes

Here, we include a hyperparameter sweep on the continuous Colonel Blotto game, as described in §5, with higher action dimensionalities and support sizes. The configuration used for this sweep is as follows.

```
def get_configs_blotto():
    return [
        {
            "game": "blotto",  # game to use
            "items": items,  # number of items / battlefields
            "soft": 1,  # smooth the utilities for the BR steps
            "softness": 0.01,  # temperature of the smoothing
            "lr": lr,  # learning rate (stepsize)
            "trials": 16,  # number of trials
            "size": size,  # size of the support set for each player
            "time_limit": 600,  # seconds of training per run
            "br_points": 100_000,  # number of discretized actions for
              ↪ best response estimation
        }
        for items in [3, 6, 12, 24]
        for lr in [1e-4, 1e-5]
        for size in [16, 32, 64, 128, 256]
    ]
```

This sweep includes the following features:

- High action dimensionalities (labeled "`items`").
- High support sizes (labeled "`size`").
- A high number of discretization points for the best response and exploitability estimation (labeled "`br_points`").
- A low temperature $\tau = 0.01$ for the utility smoothing (labeled "`softness`").

Table 2 contains the final exploitabilities, optimizing across the learning rate. The table is sorted in decreasing order of final mean exploitability. Since every run trains for 10 minutes, the table shows the number of iterations completed rather than the runtime. For 24 items, a 10^5-point discretization of the 23-dimensional simplex is relatively sparse, so the exploitability estimates (which are lower bounds; see §5) are less tight.

Table 2: Final exploitabilities on continuous Colonel Blotto for each support size and number of items, using the best learning rate (LR) for each.

			Exploitability			
Size	Items	LR	Mean	SEM	Std	Iterations
16	24	10^{-4}	3.73	0.115	0.444	6261700
32	24	10^{-4}	3.56	0.0945	0.366	4387200
64	24	10^{-5}	3.38	0.0965	0.374	2679100
128	24	10^{-5}	2.83	0.117	0.454	595100
256	24	10^{-5}	2.10	0.0545	0.211	173100
16	12	10^{-4}	1.77	0.0756	0.293	7207000
32	12	10^{-5}	1.33	0.0324	0.126	5458300
64	12	10^{-5}	1.13	0.0338	0.131	3898700
128	12	10^{-5}	1.05	0.0260	0.101	1174300
16	6	10^{-5}	0.941	0.0265	0.103	7781600
256	12	10^{-5}	0.819	0.0181	0.0703	324300
16	3	10^{-4}	0.582	0.0225	0.0873	8756600
32	6	10^{-5}	0.571	0.0107	0.0415	6126500
64	6	10^{-5}	0.445	0.00741	0.0287	4806200
128	6	10^{-5}	0.362	0.00513	0.0199	2726100
32	3	10^{-5}	0.320	0.0137	0.0531	6761500
256	6	10^{-5}	0.285	0.00447	0.0173	584600
64	3	10^{-5}	0.180	0.00446	0.0173	5638600
128	3	10^{-5}	0.115	0.00168	0.00651	3727000
256	3	10^{-5}	0.100	0.00151	0.00586	1103700

B.2 Direct Comparison to MO and RFP on Continuous Colonel Blotto

We directly compare the following algorithms on the continuous Colonel Blotto game:

- **MO**: Multiple Oracle [59]. Needs an NE and a BR oracle.
- **RFP**: Redundant Fictitious Play [36]. Needs a BR oracle.
- **IMO**: Incremental Multiple Oracle (ours). Needs no oracle.

We use CVXOPT [5] to solve the required LPs (for NE), and the HiGHS solver [51], through SciPy [116], to solve the required MILPs (for BRs). Each MILP solve has a time limit of 5 seconds, after which the best allocation found so far (or a greedy one, if that is better) is used. MO and RFP solve the MILPs of their 16 trials in parallel, on one CPU per trial.

The final exploitabilities are shown in Table 3. Its IMO rows are the results in Table 2.

Table 3: Final exploitabilities on continuous Colonel Blotto after 10 minutes (optimizing across the learning rate for IMO). Rows are sorted in decreasing order of final mean exploitability.

				Exploitability			
Algorithm	Size	Items	LR	Mean	SEM	Std	Iterations
RFP	–	24	–	16.9	0.0528	0.204	74
MO	–	24	–	9.27	0.599	2.32	98
RFP	–	12	–	7.72	0.0258	0.100	107
MO	–	12	–	3.83	0.141	0.547	135
IMO	16	24	10^{-4}	3.73	0.115	0.444	6261700
IMO	32	24	10^{-4}	3.56	0.0945	0.366	4387200
IMO	64	24	10^{-5}	3.38	0.0965	0.374	2679100
RFP	–	6	–	2.93	0.0243	0.0941	148
IMO	128	24	10^{-5}	2.83	0.117	0.454	595100
IMO	256	24	10^{-5}	2.10	0.0545	0.211	173100
IMO	16	12	10^{-4}	1.77	0.0756	0.293	7207000
IMO	32	12	10^{-5}	1.33	0.0324	0.126	5458300
IMO	64	12	10^{-5}	1.13	0.0338	0.131	3898700
IMO	128	12	10^{-5}	1.05	0.0260	0.101	1174300
IMO	16	6	10^{-5}	0.941	0.0265	0.103	7781600
IMO	256	12	10^{-5}	0.819	0.0181	0.0703	324300
IMO	16	3	10^{-4}	0.582	0.0225	0.0873	8756600
IMO	32	6	10^{-5}	0.571	0.0107	0.0415	6126500
RFP	–	3	–	0.533	0.0121	0.0469	245
MO	–	6	–	0.522	0.0120	0.0465	208
IMO	64	6	10^{-5}	0.445	0.00741	0.0287	4806200
IMO	128	6	10^{-5}	0.362	0.00513	0.0199	2726100
IMO	32	3	10^{-5}	0.320	0.0137	0.0531	6761500
IMO	256	6	10^{-5}	0.285	0.00447	0.0173	584600
IMO	64	3	10^{-5}	0.180	0.00446	0.0173	5638600
IMO	128	3	10^{-5}	0.115	0.00168	0.00651	3727000
MO	–	3	–	0.105	0.00202	0.00781	263
IMO	256	3	10^{-5}	0.100	0.00151	0.00586	1103700

Each iteration of MO and RFP solves MILPs whose size grows with the number of items, so with more items, fewer iterations complete within the 10 minutes: from 263 for 3 items to 98 for 24 items for MO, and from 245 to 74 for RFP.

With 12 or 24 items, our method attains lower final exploitability than MO and RFP at every support size. With 6 items, it does so with support sizes of 64 or more. With 3 items, MO does better than our method with support sizes up to 128, and comparably with 256; RFP does worse than our method with support sizes of 32 or more.

Moreover, the action sets of MO and RFP grow over time, which eventually causes significant slowdowns due to the need to solve larger and larger MILPs to compute the best responses. As noted by Ganzfried [36, p. 7]:

> Note that as we run successive iterations of [RFP], the size of these optimization problems becomes larger, since the opponent's strategy is a mixture over t pure strategies, where t is the current algorithm iteration. We have seen that the number of variables and constraints scales linearly in t. Therefore, we expect earlier iterations of the algorithm to run significantly faster than later iterations.

Ganzfried [36, p. 9] also reports:

> It took slightly under 25,000 s (around 6.9 h) to run 5000 iterations of our algorithm. The final strategies had an exploitability of 0.0307 for player 1 and 0.0292 for player 2.

However, these numbers are for a different game, with imperfect information, unequal budgets, and unequal battlefield values, and they measure exploitability as the maximum over the two players rather than the sum. Therefore, they are not directly comparable to the results in Table 3.

B.3 NE/BR Oracle Ablation on Continuous Colonel Blotto

Here, we test replacing the components of our algorithm with exact NE and/or BR oracles for 2-player, 3-item continuous Colonel Blotto. The oracles are those of Appendix B.2, and each run stops after 10^5 iterations or 10 minutes, whichever comes first.

Results are shown in Table 4, optimized over the learning rate. Within each support size, rows are sorted in decreasing order of final mean exploitability. All runs that use an oracle hit the time limit, whereas those without one complete 10^5 iterations in 8 to 13 seconds. Runtimes are shown in seconds. Mean, SEM, and Std are as described at the start of this appendix; the remaining columns are as follows.

- **Size** is the size of each player's support.
- **Exact BR** indicates whether an exact BR oracle was used (to update the best-performing pure strategies).
- **Exact NE** indicates whether an exact NE oracle was used (to solve the metagame).
- **Iterations** is the number of iterations completed.

Table 4: Results for each size, by whether BR and NE are computed exactly. Within each size, rows are sorted in decreasing order of mean exploitability.

			Exploitability				
Size	Exact BR	Exact NE	Mean	SEM	Std	Runtime (s)	Iterations
8	Yes	Yes	1.31	0.104	0.403	600.0	5538
	No	Yes	1.02	0.0381	0.147	600.0	68786
	Yes	No	0.967	0.0256	0.0993	600.0	4811
	No	No	0.927	0.0209	0.0810	7.7	100000
16	Yes	Yes	0.853	0.0284	0.110	600.2	2230
	Yes	No	0.791	0.0225	0.0872	600.2	1976
	No	Yes	0.647	0.0211	0.0817	600.0	60279
	No	No	0.564	0.0262	0.102	8.5	100000
32	Yes	No	0.626	0.0161	0.0624	600.1	1282
	Yes	Yes	0.624	0.0293	0.114	600.1	1073
	No	Yes	0.383	0.0123	0.0478	600.0	45843
	No	No	0.349	0.0158	0.0612	10.7	100000
64	Yes	No	0.524	0.0107	0.0415	600.6	829
	Yes	Yes	0.453	0.0117	0.0455	600.1	665
	No	Yes	0.234	0.00475	0.0184	600.0	28109
	No	No	0.188	0.00308	0.0119	12.6	100000

Across all sizes, our oracle-free algorithm attains the lowest mean exploitability. Furthermore, it does so one to two orders of magnitude faster than the alternatives, in terms of wall-clock time.

B.4 Strategy Profiles

In this subsection, we include illustrations of strategy profiles for the various games described in §5.

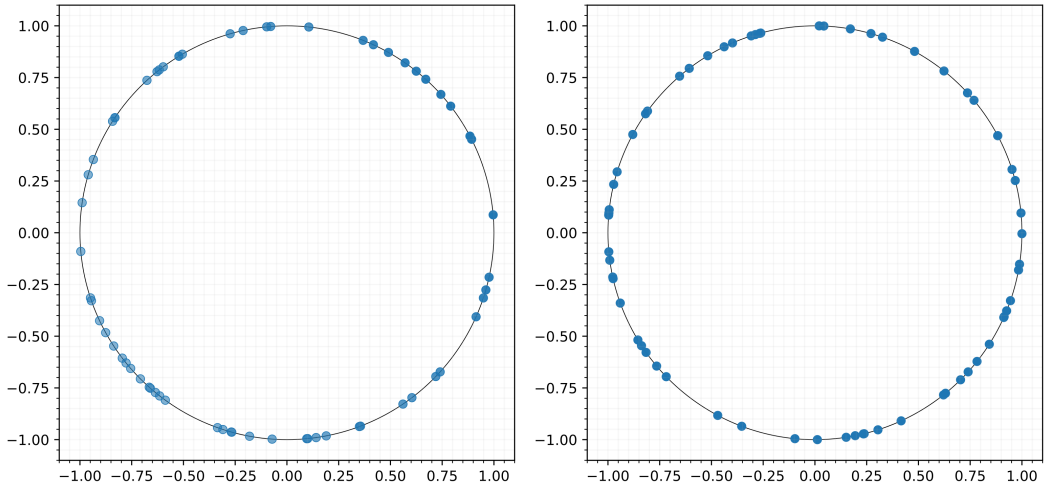


Figure 12: Example of learned strategies for the circle game, for Players 1 and 2. Each dot shows an action and its opacity shows its probability.

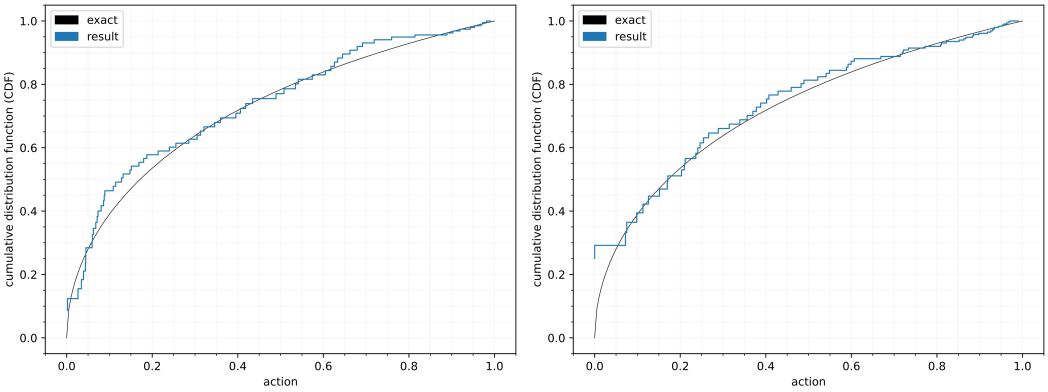


Figure 13: Example of learned strategies for the Glicksberg–Gross game, for Players 1 and 2. The exact equilibrium strategies are shown in black.

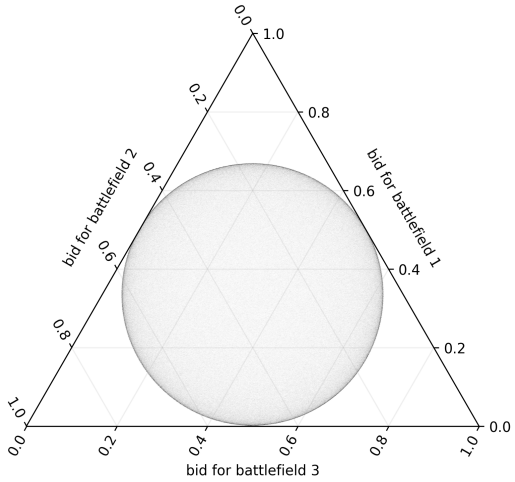


Figure 14: An exact equilibrium for the 2-player, 3-item continuous Colonel Blotto game, derived by Gross and Wagner [45]. Darkness shows density.

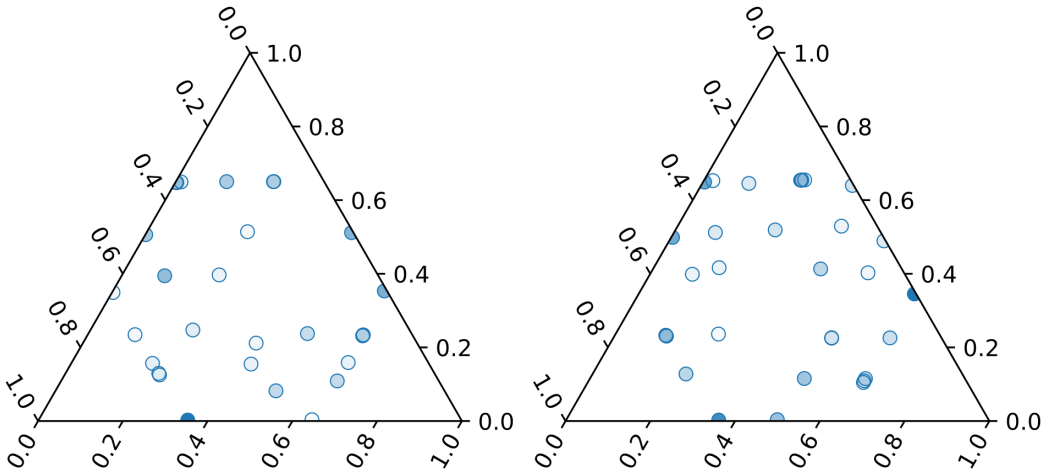


Figure 15: Example of learned strategies for the 2-player, 3-item continuous Colonel Blotto game, for Players 1 and 2. Each dot shows an action and its opacity shows its probability.

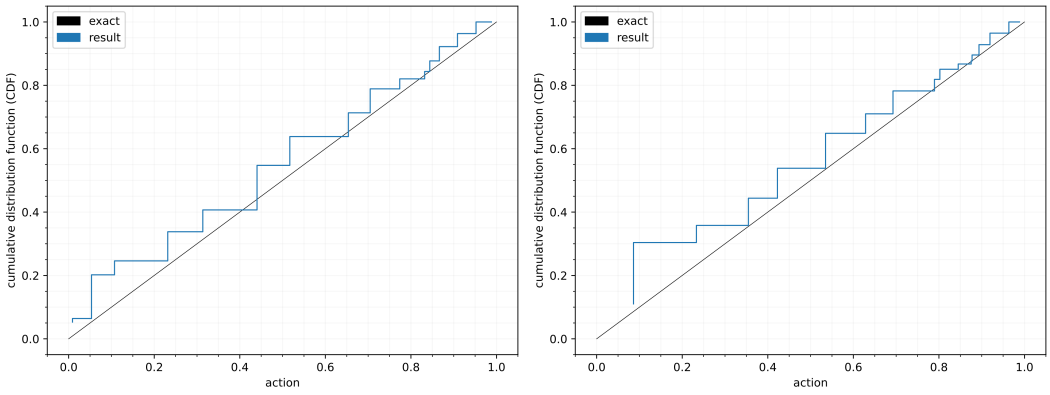


Figure 16: Example of learned strategies for the 2-player complete-information all-pay auction, for Players 1 and 2. The exact equilibrium strategies are shown in black.

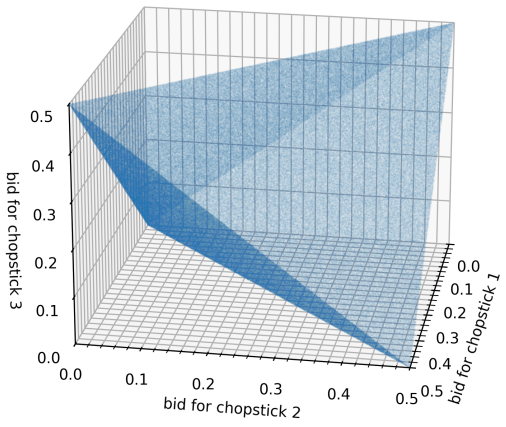


Figure 17: An exact equilibrium for the chopstick auction, derived by Szentes and Rosenthal [110, 111].

C Cost Measurements

In this section, we include some measurements of the execution cost for different support sizes, in terms of time (per iteration), floating-point operations (flops), and memory (bytes accessed). Time was obtained by dividing the total measured wall time of the experiment by the total number of iterations. Flops and memory were measured via JAX's cost analysis tools. Results are shown in the following figures, in each of which the bars are sorted by value rather than by support size. In this section and in Appendix D, the circle game is a variant in which each player chooses a point of $[0, 1)$, with the endpoints identified, and $u_1(x, y) = -u_2(x, y) = \min(|x - y|, 1 - |x - y|)$, which is the distance along a circle of circumference 1.

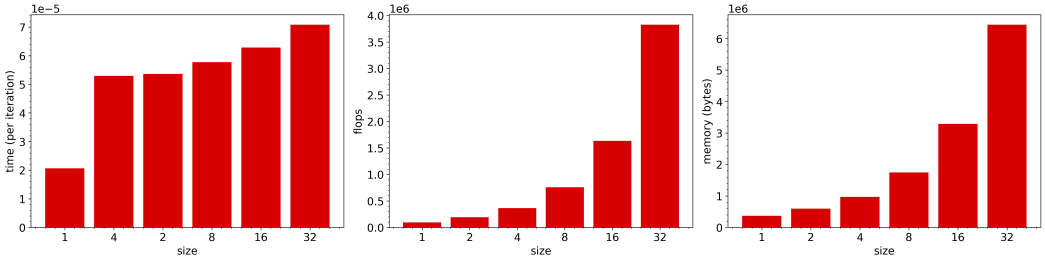


Figure 18: Costs for the interval game.

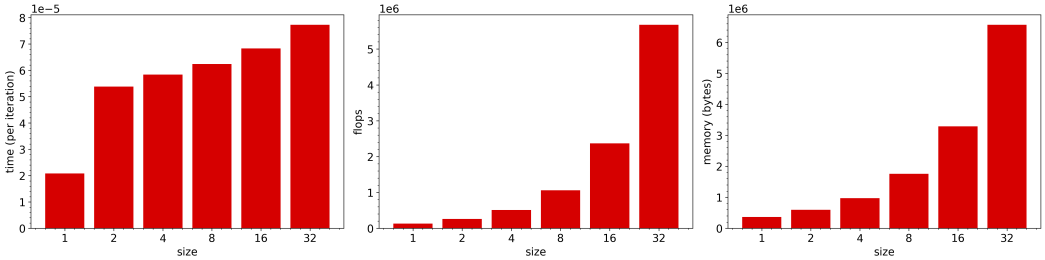


Figure 19: Costs for the circle game.

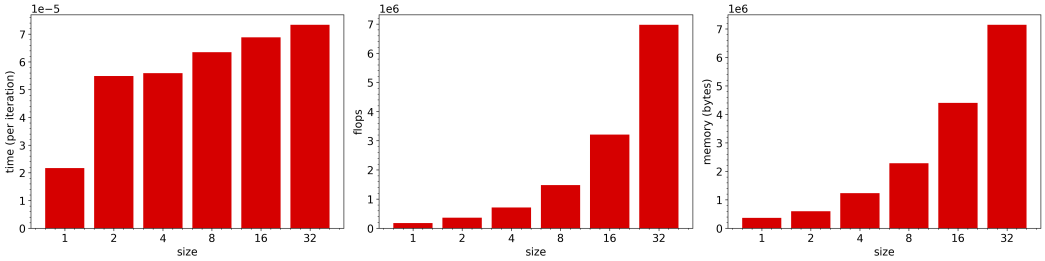


Figure 20: Costs for the Glicksberg–Gross game.

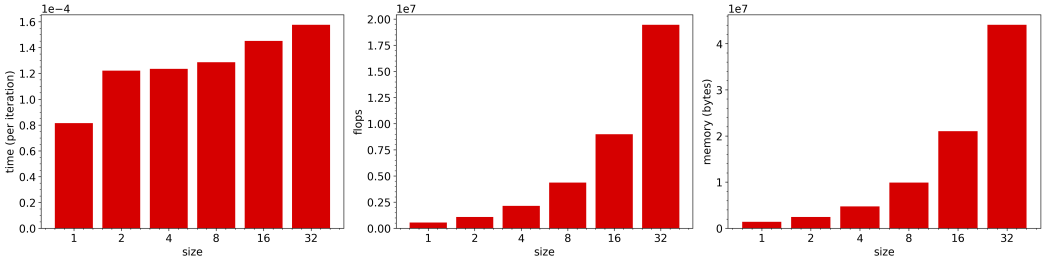


Figure 21: Costs for the continuous Colonel Blotto game.

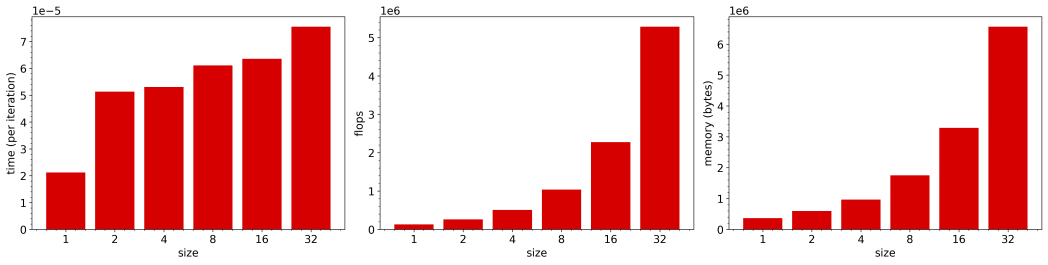


Figure 22: Costs for the 1D security game.

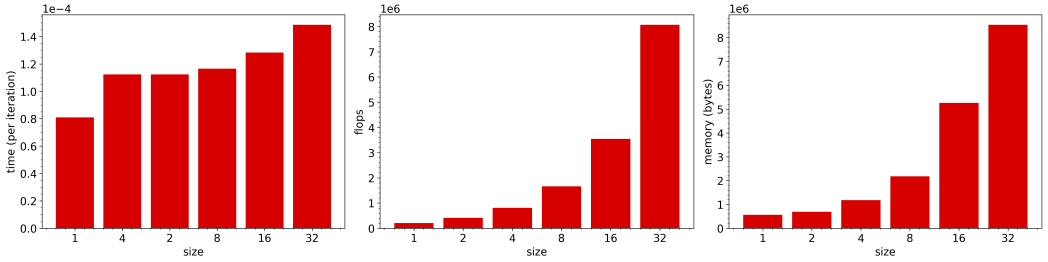


Figure 23: Costs for the 2D security game.

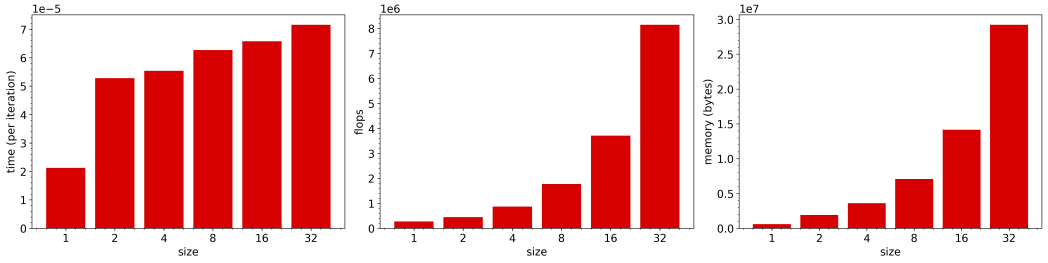


Figure 24: Costs for the 2-player all-pay auction.

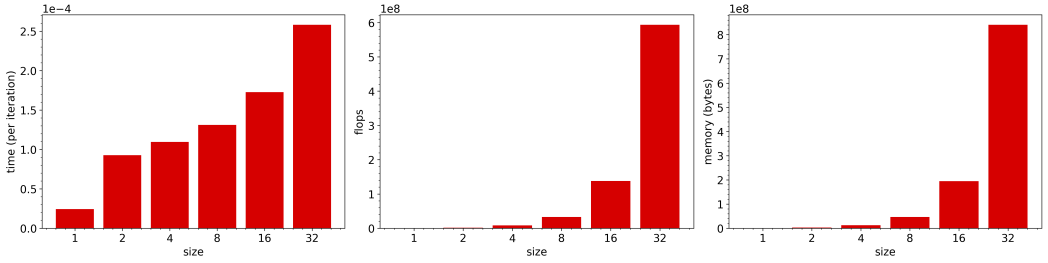


Figure 25: Costs for the 3-player all-pay auction.

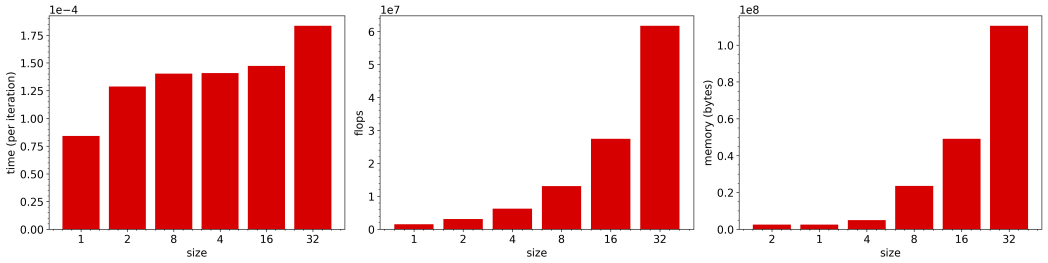


Figure 26: Costs for the chopstick auction.

D Tradeoff in Sub-iterations

Our algorithm has the advantage over MO of being fully *gradual*. On each iteration, it can do a minimum amount of work on each of the metagame-solving and pure strategy exploration fronts. The same is not true for the usual approach where the cost of solving the metagame is fixed. This presents a tradeoff between spending more time on solving the metagame and on exploring the pure strategy space (i.e., adjusting the pure strategies). Specifically, our algorithm has K_{ED}, which is the number of sub-iterations used to perform ED in the metagame, and K_{BR}, which is the number of sub-iterations used to move around the supports. Letting $K_{\text{ED}} \to \infty$ corresponds to (approximately) solving the metagame on each iteration, as the metagame solvers of MO and PSRO do, whereas letting $K_{\text{BR}} \to \infty$ corresponds to moving each support strategy to a (local) best response on each iteration, analogous to the BR oracles of MO and PSRO (but without adding new strategies). In most of our experiments, we considered the simplest case where $K_{\text{ED}} = K_{\text{BR}} = 1$. In this section, we perform some experiments to measure what happens when we use different ratios. Each of these runs uses 2×10^5 iterations. Results are shown in the following figures. In the plot legends, "meta_steps" denotes K_{ED}, and "support_steps" denotes K_{BR}. The circle game here is the variant described in Appendix C. The relative benefit of each tradeoff appears to be environment-dependent. Also, notably, exploitability still decreases to some extent even when $K_{\text{ED}} = 0$ or $K_{\text{BR}} = 0$. However, the final exploitability for such settings is generally significantly higher than for the others. The one exception is the setting $K_{\text{ED}} = 0, K_{\text{BR}} = 1$ in the Colonel Blotto game, which eventually performs comparably to the settings in which both are positive. Overall, this ablation shows that dynamic adjustment of both metagame probabilities and supports is beneficial.

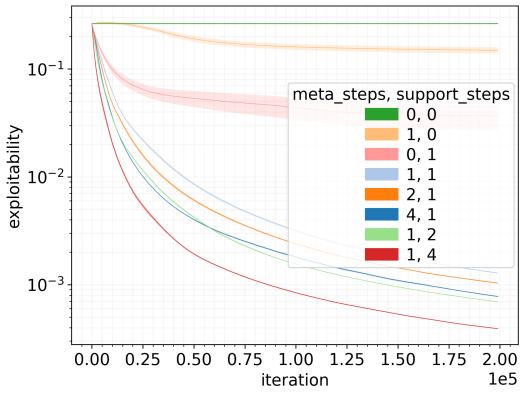


Figure 27: Sub-iteration tradeoff for the interval game.

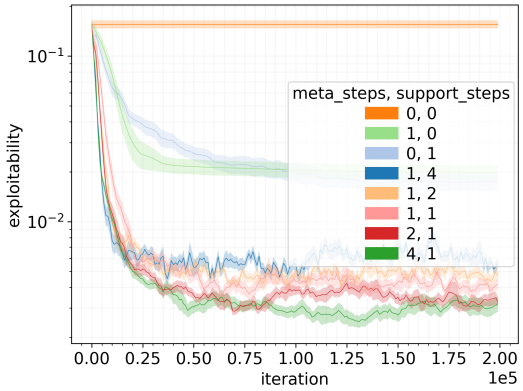


Figure 28: Sub-iteration tradeoff for the circle game.

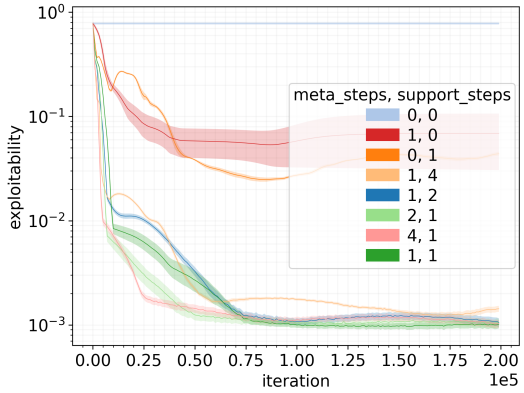


Figure 29: Sub-iteration tradeoff for the Glicksberg–Gross game.

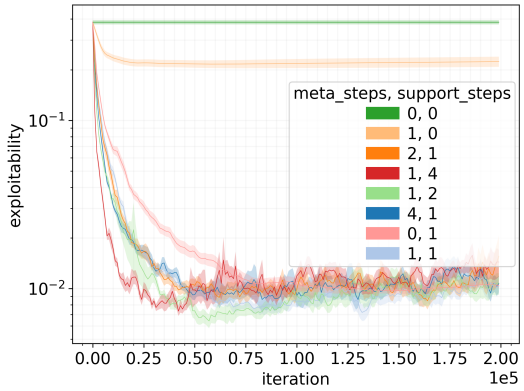


Figure 30: Sub-iteration tradeoff for the continuous Colonel Blotto game.

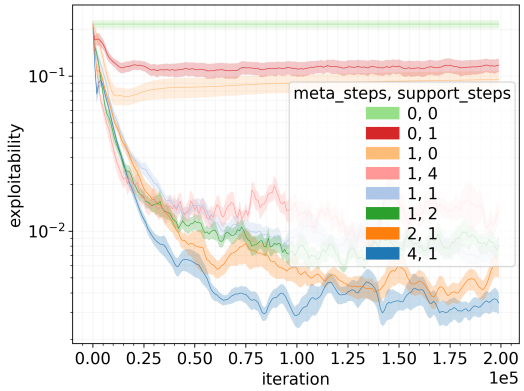


Figure 31: Sub-iteration tradeoff for the 2-player all-pay auction.

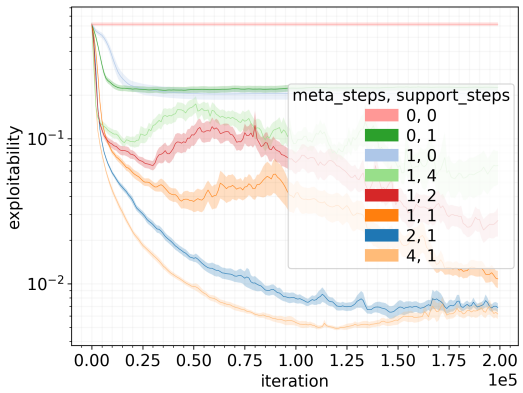


Figure 32: Sub-iteration tradeoff for the 3-player all-pay auction.

E Additional Related Research

Perkins et al. [95] proposed an actor-critic *reinforcement learning (RL)* algorithm in which each player learns a mixed strategy over a continuous action space, and proved that, in potential games, the associated continuous-time dynamics converge to equilibrium.

Bichler et al. [14] computed equilibria of Bayesian games, such as auctions, in distributional strategies [83], which generalize mixed strategies to this setting. They discretized the type and action spaces and learned such strategies for the discretized game via online convex optimization, specifically *simultaneous online dual averaging (SODA)*. They also showed that equilibria computed on the discretization approximate equilibria of the continuous game.

Bichler et al. [12] represented strategies by neural networks and trained them in self-play with simultaneous gradient dynamics, an approach that provably learns local equilibria. They focused on symmetric auctions, in which all bidders have the same prior distribution and use the same equilibrium bidding strategy. Bichler et al. [13] extended this approach to asymmetric auctions, which require training several interacting neural networks. Both papers consider only pure strategies.

Martin and Sandholm [69] tackled continuous-action Bayesian games in generality by modeling mixed strategies using randomized policy networks and applying zeroth-order optimization techniques that combine smoothed gradient estimators with equilibrium-finding dynamics.

E.1 Equilibrium Existence and Uniqueness

Nash [86] showed that if each S_i is nonempty and finite, a mixed-strategy NE exists. Glicksberg [39] showed that if each S_i is nonempty and compact, and each u_i is continuous, a mixed-strategy NE exists. Glicksberg [39], Fan [32], and Debreu [28] showed that if each S_i is nonempty, compact, and convex, and each u_i is continuous and quasiconcave in s_i, a pure-strategy NE exists. Dasgupta and Maskin [27] showed that if each S_i is nonempty, compact, and convex, and each u_i is upper semicontinuous, graph continuous, and quasiconcave in s_i, a pure-strategy NE exists. They also showed that if each S_i is nonempty, compact, and convex, and each u_i is bounded and continuous except on a subset (defined by technical conditions) and weakly lower semicontinuous in s_i, and $\sum_{i \in I} u_i$ is upper semicontinuous, a mixed-strategy NE exists. Rosen [103] proved the uniqueness of a pure NE for continuous-action games under diagonal strict concavity assumptions.

E.2 Abstraction

Brown and Sandholm [22] considered optimizing a player's parameter vector in a two-player zero-sum game, such as the bet sizes available to a poker player. Their gradient-descent algorithm provably reaches a local optimum of the parameters while computing an equilibrium at the same time. This made it the first action abstraction algorithm (i.e., a method for choosing a few discrete actions from a continuum, an important preprocessing step when solving large games with current equilibrium-finding algorithms) that is guaranteed to converge in extensive-form games.

Abstraction plays a central role in solving incomplete-information extensive-form games. Kroer and Sandholm [56] developed a theoretical framework for bounding the quality of solutions computed in abstractions of any perfect-recall extensive-form game, based on a new way of mapping abstract strategies back to the original game and on a new equilibrium refinement. With it, they obtained the first general lossy abstraction method for extensive-form games with solution-quality bounds.

Many real-world settings have continuous action spaces, which are usually discretized heuristically, without any guarantee on solution quality. Building on earlier results about the quality of abstractions, Kroer and Sandholm [57] gave the first framework that provides solution-quality bounds when discretizing the continuous action spaces of extensive-form games.

Brown and Sandholm [23] interleaved abstraction with equilibrium finding: the agent starts from a coarse abstraction and adds actions at run time wherever the strategy computed so far deems them important, provably without having to restart equilibrium finding. This yields anytime convergence to an NE of the full game, which may even be infinite.

Kroer and Sandholm [58] gave the first general solution-quality guarantees, independent of the equilibrium-finding algorithm, for NE and approximate self-trembling equilibria that are computed in imperfect-recall abstractions and then played in the original perfect-recall game.

E.3 Fictitious Play

Brown [21] introduced *fictitious play (FP)*, a widely studied model of how players learn in games. In FP, a game is played repeatedly, and on each iteration, each player best responds to the average of its opponents' past strategies. In two-player zero-sum games, potential games, and some other classes of games, the players' average strategies converge to the set of NE. The players may update their beliefs simultaneously or in alternation [10]. Leslie and Collins [62] introduced *generalised weakened FP (GWFP)*, a generalization of FP that tolerates approximate best responses and perturbed updates of the average strategies.

Perkins and Leslie [94] extended stochastic FP to continuous action spaces. By studying its limiting behavior through smooth best response dynamics over finite signed measures, they showed that it converges to an equilibrium in two-player zero-sum games.

Heinrich et al. [49] introduced *full-width extensive-form FP (XFP)*, a version of FP for extensive-form (multi-step) games that operates on behavioral strategies. They also introduced *fictitious self-play (FSP)*, which realizes GWFP in behavioral strategies through machine learning from sampled experience. FSP players play the game repeatedly and keep their experience in memory. Rather than playing a best response outright, each player cautiously mixes its best response with its average strategy. On each iteration, each player learns an approximate best response from its stored experience of playing against its opponents, and learns a model of its own average strategy from the record of its own past behavior.

Heinrich and Silver [48] introduced *neural fictitious self-play (NFSP)*, which combines FSP with deep reinforcement learning. Each NFSP agent has two neural networks trained on stored experience: one uses reinforcement learning to approximate a best response to the other agents' past behavior, and the other uses supervised learning to imitate the agent's own average behavior.

Kamra et al. [55] introduced *DeepFP*, which approximately implements FP in continuous-action two-player games. It represents approximate best responses with generative neural networks (expressive implicit density models) and learns a differentiable model of the players' payoffs as a function of their actions, training all networks jointly in a model-based manner. When domain-specific best-response oracles are available, they can replace the best-response networks, so that gradients are not needed. DeepFP converged stably to NE in several classic games, and it was also applied to a large-scale forest security game.

Ganzfried [36] introduced *redundant FP (RFP)*, an algorithm for approximating equilibria in continuous-action games, and applied it to a continuous Colonel Blotto game. Unlike ours, this algorithm requires a best-response oracle as a subroutine (e.g., a mixed-integer linear program solver for the continuous Colonel Blotto game).

Li and Wellman [63] presented an equilibrium-finding algorithm for symmetric n-player general-sum continuous Bayesian games. It represents strategies by neural networks and trains them with *natural evolution strategies (NES)* [120]. To approximate a pure-strategy equilibrium, it solves a bilevel optimization problem, using NES both to compute BRs in the inner loop and to minimize regret in the outer loop.

E.4 Policy-Space Response Oracles (PSRO)

Lanctot et al. [61] introduced *policy-space response oracles (PSRO)*. It extends DO to metagames whose strategies are policies instead of actions, and it also generalizes FSP. Unlike earlier methods, it can use any meta-solver to compute the meta-strategies.

McAleer et al. [72] introduced *pipeline PSRO (P2SRO)*, which scales PSRO to large zero-sum imperfect-information games. It parallelizes PSRO while retaining convergence guarantees by arranging reinforcement learning workers in a hierarchical pipeline, in which each worker trains against the policies produced by the levels below it.

McAleer et al. [73] introduced *extensive-form DO (XDO)*, a DO algorithm for two-player zero-sum extensive-form games that provably reaches an approximate NE within a number of iterations that grows linearly with the number of infostates. PSRO mixes best responses only at the root node of the game tree, whereas XDO mixes them at every infostate. They also introduced *neural XDO (NXDO)*, which learns best responses with deep RL and computes approximate equilibria of its restricted games with a deep RL equilibrium solver (e.g., NFSP [48] or DREAM [109]). Like PSRO, NXDO grows a population of reinforcement learning policies, but it solves an extensive-form restricted game rather than a matrix-form one, which they showed to be more efficient.

McAleer et al. [75] introduced *anytime DO (ADO)*, a tabular variant of DO for *two-player zero-sum* games that provably converges to an NE and whose exploitability, unlike that of DO, never increases across iterations. They also introduced *anytime PSRO (APSRO)*, which is ADO with best responses computed by reinforcement learning.

McAleer et al. [74] introduced *self-play PSRO (SP-PSRO)*. Earlier PSRO variants grow the population only with deterministic policies that best respond to the least-exploitable mixture of the opponent's population, so they may have to include every deterministic policy before they converge. On each iteration, SP-PSRO additionally trains a stochastic policy that is approximately optimal and inserts it into the population. However, as its authors note, SP-PSRO operates in normal form, mixing policies only at the root node of the game tree. McAleer et al. [73] showed that such methods may need exponentially many iterations (in the number of infostates) to converge to an approximate NE, and introduced XDO and NXDO to address this. McAleer et al. [74] leave it to future work to combine XDO and NXDO with SP-PSRO. Like APSRO, SP-PSRO is limited to two-player zero-sum games.

Muller et al. [84] developed the theory of PSRO with α-Rank [89], instead of NE, as the solution concept. They proved convergence guarantees for several classes of games and related α-Rank to NE.

Marris et al. [68] proposed *joint policy-space response oracles (JPSRO)*, which trains agents in general-sum extensive-form games with any number of players. Instead of NE, it targets *correlated equilibrium (CE)* [6] and *coarse correlated equilibrium (CCE)*.

These methods are also related to the broader class of coevolutionary algorithms [33, 34, 88], in which strategically interacting agents adapt to one another, and the outcomes of their interactions steer evolution toward increasingly adaptive behaviors.

F Code

The following is an example implementation of our method, applied to the Glicksberg–Gross game. For our experiments, we used Python 3.12.3 [114], JAX 0.7.0 [20], Flax 0.8.5 [47], Optax 0.2.5 [29], Matplotlib 3.10.6 [52], and SciPy 1.16.1 [116], except for the experiments in §5, which we ran in 2024 with the versions of these libraries available then. For the tables in Appendix B, we used Python 3.14.8, JAX 0.12.0.1, Optax 0.2.8, SoftJAX 0.1.1 [93], SciPy 1.18.0 (whose MILP solver is HiGHS [51]), and CVXOPT 1.3.3 [5]. The output of this code is shown in Figure 33.

```python
import argparse

import jax
import optax
from jax import lax, nn, random
from jax import numpy as jnp
from jax.scipy.stats import rankdata
from matplotlib import pyplot as plt


def delete(lst, index):
  return lst[:index] + lst[index:][1:]


def replace(lst, index, item):
  return lst[:index] + [item] + lst[index:][1:]


def parse_args():
  p = argparse.ArgumentParser()
  p.add_argument("--game", default="glicksberg_gross")
  p.add_argument("--sizes", type=int, default=[31, 32], nargs="*")

  p.add_argument("--freeze_support", type=int, default=0)
  p.add_argument("--freeze_weight", type=int, default=0)

  p.add_argument("--opt", default="sgd")
  p.add_argument("--lr", type=float, default=1e-3)

  p.add_argument("--seed", type=int, default=0)
  p.add_argument("--iters", type=int, default=10**6)
  return p.parse_args()


def create_nfg(f, supports):
  in_axes = [None] * len(supports)
  for i in reversed(range(len(supports))):
    f = jax.vmap(f, [replace(in_axes, i, 0)])
  tensor = f(supports)
  assert tensor.shape == (*map(len, supports), len(supports))
  return jnp.moveaxis(tensor, -1, 0)


def get_nfg_utilities(utilities, strategies):
  for strategy in reversed(strategies):
    utilities @= strategy
  return utilities


def get_nfg_player_utilities(utilities, strategies, player):
  utilities = utilities[player]
  utilities = jnp.moveaxis(utilities, player, 0)
  for strat in reversed(delete(strategies, player)):
    utilities @= strat
  return utilities
```

```
def get_nfg_exploitability(utilities, strategies):
  br_values = [
    get_nfg_player_utilities(utilities, strategies, player).max()
    for player in range(len(strategies))
  ]
  current_utilities = get_nfg_utilities(utilities, strategies)
  gaps = jnp.stack(br_values) - current_utilities
  return gaps.sum()


def mix(x, axis=-1):
  ranks = rankdata(x, method="ordinal", axis=axis)
  weights = ranks / ranks.shape[axis]
  return jnp.vecdot(x, weights, axis=axis)


def init_support_params(space, size):
  # Evenly spaced actions, excluding the endpoints (where the inverse is
  ↪ infinite).
  actions = space.discretize(size + 2)[1:-1]
  return space.reparameterize_inverse(actions)


def create_state(key, sizes, game, optimizer):
  support_params = [
    init_support_params(game.action_space(player), size)
    for player, size in enumerate(sizes)
  ]
  weight_params = [jnp.zeros(size) for size in sizes]
  params = {"support": support_params, "weight": weight_params}
  opt_state = optimizer.init(params)
  return {"params": params, "opt_state": opt_state}


class Interval:
  def sample(self, key):
    return random.uniform(key)

  def discretize(self, points):
    return jnp.linspace(0, 1, points)

  def reparameterize(self, x):
    return nn.sigmoid(x)

  def reparameterize_inverse(self, x):
    return jax.scipy.special.logit(x)

  def shape(self):
    return ()


class Simplex:
  def __init__(self, n):
    self.n = n

  def sample(self, key):
    x = random.exponential(key, [self.n])
    return x / x.sum()

  def reparameterize(self, x):
    return nn.softmax(x)

  def shape(self):
```

```
    return (self.n,)


class GlicksbergGross:
  """https://www.degruyter.com/document/doi/10.1515/9781400881970-010"""

  def action_space(self, player):
    return Interval()

  def utilities(self, actions):
    x, y = actions
    z = (1 + x) * (1 + y) * (1 - x * y) / jnp.square(1 + x * y)
    return jnp.stack([z, -z])

  def plot_strategy(self, player, support, weight):
    i = support.argsort()

    fig, ax = plt.subplots()
    ax.step(support[i], weight[i].cumsum(), label="learned")

    ax.set(
      title=f"player {player + 1}",
      xlabel="action",
      ylabel="cumulative probability",
    )

    x = jnp.linspace(0, 1)
    y = 4 / jnp.pi * jnp.arctan(jnp.sqrt(x))
    ax.plot(x, y, "k--", label="exact")
    ax.legend()


def plot_params(params, game):
  support = get_support(params, game)
  weight = get_weight(params)
  for player, (p_support, p_weight) in enumerate(zip(support, weight)):
    game.plot_strategy(player, p_support, p_weight)


def get_exploit(params, key, game):
  support = get_support(params, game)
  weight = get_weight(params)

  nfg = create_nfg(game.utilities, support)

  def get_br_utility(player):
    new_support = replace(
      support,
      player,
      game.action_space(player).discretize(100_000),
    )
    new_nfg = create_nfg(game.utilities, new_support)
    utilities = get_nfg_player_utilities(new_nfg, weight, player)
    return utilities.max()

  num_players = len(weight)
  assert num_players == len(support)

  br_utilities = [get_br_utility(player) for player in range(num_players)
    ↪ ]
  regrets = jnp.stack(br_utilities) - get_nfg_utilities(nfg, weight)
  return regrets.sum()


def get_metagame_exploit(params, key, game):
```

```
  support = get_support(params, game)
  weight = get_weight(params)
  nfg = create_nfg(game.utilities, support)
  return get_nfg_exploitability(nfg, weight)


def get_support(params, game):
  return [
    jax.vmap(game.action_space(player).reparameterize)(p)
    for player, p in enumerate(params["support"])
  ]


def get_weight(params):
  return [Simplex(p.size).reparameterize(p) for p in params["weight"]]


def get_support_params_grads(params, key, game):
  def f(new_params, old_params):
    new_support = get_support(new_params, game)
    old_support = get_support(old_params, game)
    weight = get_weight(params)

    def g(player):
      mix_support = replace(old_support, player, new_support[player])
      nfg = create_nfg(game.utilities, mix_support)
      utilities = get_nfg_player_utilities(nfg, weight, player)
      return -mix(utilities)

    num_players = len(new_support)
    return sum(map(g, range(num_players)))

  return jax.grad(f)(params, params)["support"]


def get_weight_params_grads(params, key, game):
  return jax.grad(get_metagame_exploit)(params, key, game)["weight"]


def get_metrics(params, key, game):
  return {
    "exploitability": get_exploit(params, key, game),
    "metagame exploitability": get_metagame_exploit(params, key, game),
  }


def get_game(args):
  match args.game:
    case "glicksberg_gross":
      return GlicksbergGross()
    case _:
      raise NotImplementedError


def get_optimizer(args):
  match args.opt:
    case "sgd":
      return optax.sgd(args.lr)
    case _:
      raise NotImplementedError


def main():
  args = parse_args()
  game = get_game(args)
```

```
  optimizer = get_optimizer(args)

  def apply_grads(params, opt_state, grads):
    updates, opt_state = optimizer.update(grads, opt_state)
    return optax.apply_updates(params, updates), opt_state

  def update_state(state, key):
    params, opt_state = state["params"], state["opt_state"]
    zeros = optax.tree_utils.tree_zeros_like

    # Reduce the metagame exploitability by adjusting the weights.
    if not args.freeze_weight:
      grads = {
        "support": zeros(params["support"]),
        "weight": get_weight_params_grads(params, key, game),
      }
      params, opt_state = apply_grads(params, opt_state, grads)

    # Improve the supports against the updated weights.
    if not args.freeze_support:
      grads = {
        "support": get_support_params_grads(params, key, game),
        "weight": zeros(params["weight"]),
      }
      params, opt_state = apply_grads(params, opt_state, grads)

    new_state = {"params": params, "opt_state": opt_state}
    return new_state, get_metrics(params, key, game)

  key = random.key(args.seed)

  key, subkey = random.split(key)
  init_state = create_state(subkey, args.sizes, game, optimizer)

  keys = random.split(key, args.iters)
  final_state, history = lax.scan(update_state, init_state, keys)

  plt.rcParams["figure.constrained_layout.use"] = True
  plt.rcParams["savefig.dpi"] = 300

  for metric in history:
    fig, ax = plt.subplots()
    ax.plot(history[metric])
    ax.set(xlabel="iteration", ylabel=metric, yscale="log")

  plot_params(final_state["params"], game)

  plt.show()


if __name__ == "__main__":
  main()
```

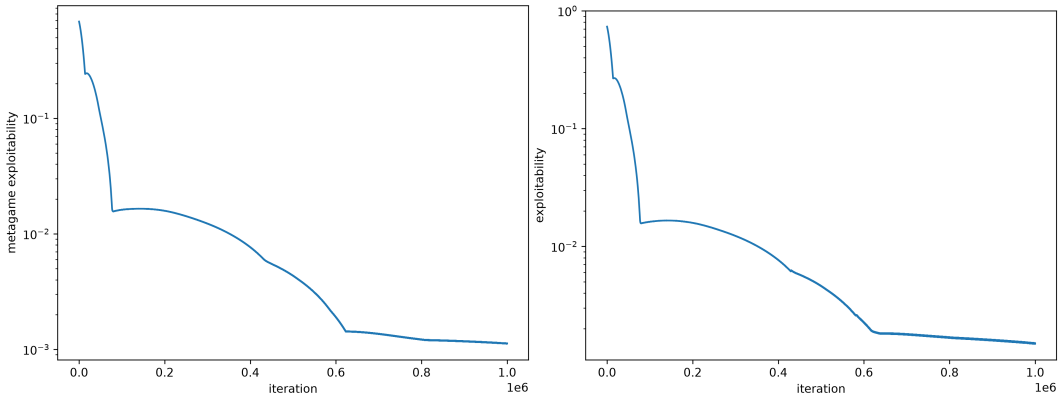


Figure 33: Output of the example code on the Glicksberg–Gross game: exploitability in the metagame (left) and in the full game (right) over the course of training.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and §1 state the contributions, which are supported by the method in §4 and the experiments in §5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: §6 discusses limitations, including the lack of a general convergence guarantee, the dependence of solution quality on the support size, and the measurement of exploitability in §5 on the smoothed versions of games with discontinuous utilities.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not present formal theorems; Appendix A contains an exploitability decomposition (with its derivation) and pointers to existing guarantees.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Algorithm 2 and §4 describe the method, §5 describes the games and hyperparameters, and Appendix F contains example code.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Appendix F contains an example implementation of our method. The experiments do not use any data.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: §5 specifies the hyperparameters, including the learning rates, support sizes, and numbers of iterations, and Appendix B gives the configuration of the larger Colonel Blotto sweep.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: As described in §5, plots show the mean across 16 trials, with bands showing the standard error of the mean; the appendix tables also report standard deviations.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: §5 describes the hardware, and Appendix C and the appendix tables report execution times.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This is foundational research on computing equilibria in games. Such games can model settings including auctions, security, and defense (§1 and §5), but the paper does not target any specific deployment.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release data or models that have a high risk for misuse.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Appendix F cites the software packages that we use, along with their versions. Their licenses are as follows: Python (PSF License); JAX, Flax, Optax, and SoftJAX (Apache 2.0); Matplotlib (Matplotlib License, based on the PSF License); and SciPy (BSD 3-Clause). For the oracles in Appendices B.2 and B.3, we also use CVXOPT (GPL 3.0 or later) and the HiGHS solver included in SciPy (MIT).

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.

- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development in this research does not involve LLMs.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.