Pipeline PSRO: A Scalable Approach for Finding Approximate Nash Equilibria in Large Games

Stephen McAleer*
Department of Computer Science
University of California, Irvine
Irvine, CA
smcaleer@uci.edu

John Lanier*
Department of Computer Science
University of California, Irvine
Irvine, CA
jblanier@uci.edu

Roy Fox
Department of Computer Science
University of California, Irvine
Irvine, CA
royf@uci.edu

Pierre Baldi
Department of Computer Science
University of California, Irvine
Irvine, CA
pfbaldi@ics.uci.edu

Abstract

Finding approximate Nash equilibria in zero-sum imperfect-information games is challenging when the number of information states is large. Policy Space Response Oracles (PSRO) is a deep reinforcement learning algorithm grounded in game theory that is guaranteed to converge to an approximate Nash equilibrium. However, PSRO requires training a reinforcement learning policy at each iteration, making it too slow for large games. We show through counterexamples and experiments that DCH and Rectified PSRO, two existing approaches to scaling up PSRO, fail to converge even in small games. We introduce Pipeline PSRO (P2SRO), the first scalable PSRO-based method for finding approximate Nash equilibria in large zero-sum imperfect-information games. P2SRO is able to parallelize PSRO with convergence guarantees by maintaining a hierarchical pipeline of reinforcement learning workers, each training against the policies generated by lower levels in the hierarchy. We show that unlike existing methods, P2SRO converges to an approximate Nash equilibrium, and does so faster as the number of parallel workers increases, across a variety of imperfect information games. We also introduce an open-source environment for Barrage Stratego, a variant of Stratego with an approximate game tree complexity of 10^{50}. P2SRO is able to achieve state-of-the-art performance on Barrage Stratego and beats all existing bots. Experiment code is available at https://github.com/JBLanier/pipeline-psro.

1 Introduction

A long-standing goal in artificial intelligence and algorithmic game theory has been to develop a general algorithm which is capable of finding approximate Nash equilibria in large imperfect-information two-player zero-sum games. AlphaStar [Vinyals et al., 2019] and OpenAI Five [Berner et al., 2019] were able to demonstrate that variants of self-play reinforcement learning are capable of achieving expert-level performance in large imperfect-information video games. However, these methods are not principled from a game-theoretic point of view and are not guaranteed to converge to an approximate Nash equilibrium. Policy Space Response Oracles (PSRO) [Lanctot et al., 2017]

*Authors contributed equally

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

is a game-theoretic reinforcement learning algorithm based on the Double Oracle algorithm and is guaranteed to converge to an approximate Nash equilibrium.

PSRO is a general, principled method for finding approximate Nash equilibria, but it may not scale to large games because it is a sequential algorithm that uses reinforcement learning to train a full best response at every iteration. Two existing approaches parallelize PSRO: Deep Cognitive Hierarchies (DCH) [Lanctot et al., 2017] and Rectified PSRO [Balduzzi et al., 2019], but both have counterexamples on which they fail to converge to an approximate Nash equilibrium, and as we show in our experiments, neither reliably converges in random normal form games.

Although DCH approximates PSRO, it has two main limitations. First, DCH needs the same number of parallel workers as the number of best response iterations that PSRO takes. For large games, this requires a very large number of parallel reinforcement learning workers. This also requires guessing how many iterations the algorithm will need before training starts. Second, DCH keeps training policies even after they have plateaued. This introduces variance by allowing the best responses of early levels to change each iteration, causing a ripple effect of instability. We find that, in random normal form games, DCH rarely converges to an approximate Nash equilibrium even with a large number of parallel workers, unless their learning rate is carefully annealed.

Rectified PSRO is a variant of PSRO in which each learner only plays against other learners that it already beats. We prove by counterexample that Rectified PSRO is not guaranteed to converge to a Nash equilibrium. We also show that Rectified PSRO rarely converges in random normal form games.

In this paper we introduce Pipeline PSRO (P2SRO), the first scalable PSRO-based method for finding approximate Nash equilibria in large zero-sum imperfect-information games. P2SRO is able to scale up PSRO with convergence guarantees by maintaining a hierarchical pipeline of reinforcement learning workers, each training against the policies generated by lower levels in the hierarchy. P2SRO has two classes of policies: fixed and active. Active policies are trained in parallel while fixed policies are not trained anymore. Each parallel reinforcement learning worker trains an active policy in a hierarchical pipeline, training against the meta Nash equilibrium of both the fixed policies and the active policies on lower levels in the pipeline. Once the performance increase of the lowest-level active worker in the pipeline does not improve past a given threshold in a given amount of time, the policy becomes fixed, and a new active policy is added to the pipeline. P2SRO is guaranteed to converge to an approximate Nash equilibrium. Unlike Rectified PSRO and DCH, P2SRO converges to an approximate Nash equilibrium across a variety of imperfect information games such as Leduc poker and random normal form games.

We also introduce an open-source environment for Barrage Stratego, a variant of Stratego. Barrage Stratego is a large two-player zero sum imperfect information board game with an approximate game tree complexity of 10^{50}. We demonstrate that P2SRO is able to achieve state-of-the-art performance on Barrage Stratego, beating all existing bots.

To summarize, in this paper we provide the following contributions:

- We develop a method for parallelizing PSRO which is guaranteed to converge to an approximate Nash equilibrium, and show that this method outperforms existing methods on random normal form games and Leduc poker.
- We present theory analyzing the performance of PSRO as well as a counterexample where Rectified PSRO does not converge to an approximate Nash equilibrium.
- We introduce an open-source environment for Stratego and Barrage Stratego, and demonstrate state-of-the-art performance of P2SRO on Barrage Stratego.

2 Background and Related Work

A two-player normal-form game is a tuple (Π, U), where $\Pi = (\Pi_1, \Pi_2)$ is the set of policies (or strategies), one for each player, and $U : \Pi \to \mathbb{R}^2$ is a payoff table of utilities for each joint policy played by all players. For the game to be zero-sum, for any pair of policies $\pi \in \Pi$, the payoff $u_i(\pi)$ to player i must be the negative of the payoff $u_{-i}(\pi)$ to the other player, denoted $-i$. Players try to maximize their own expected utility by sampling from a distribution over the policies $\sigma_i \in \Sigma_i = \Delta(\Pi_i)$. The set of best responses to a mixed policy σ_i is defined as the set of policies

that maximally exploit the mixed policy: $\text{BR}(\sigma_i) = \arg\min_{\sigma'_{-i} \in \Sigma_{-i}} u_i(\sigma'_{-i}, \sigma_i)$, where $u_i(\sigma) = \text{E}_{\pi \sim \sigma}[u_i(\pi)]$. The exploitability of a pair of mixed policies σ is defined as: $\text{EXPLOITABILITY}(\sigma) = \frac{1}{2}(u_2(\sigma_1, \text{BR}(\sigma_1)) + u_1(\text{BR}(\sigma_2), \sigma_2)) \geq 0$. A pair of mixed policies $\sigma = (\sigma_1, \sigma_2)$ is a Nash equilibrium if $\text{EXPLOITABILITY}(\sigma) = 0$. An approximate Nash equilibrium at a given level of precision ϵ is a pair of mixed policies σ such that $\text{EXPLOITABILITY}(\sigma) \leq \epsilon$ [Shoham and Leyton-Brown, 2008].

In small normal-form games, Nash equilibria can be found via linear programming [Nisan et al., 2007]. However, this quickly becomes infeasible when the size of the game increases. In large normal-form games, no-regret algorithms such as fictitious play, replicator dynamics, and regret matching can asymptotically find approximate Nash equilibria [Fudenberg et al., 1998, Taylor and Jonker, 1978, Zinkevich et al., 2008]. Extensive form games extend normal-form games and allow for sequences of actions. Examples of perfect-information extensive form games include chess and Go, and examples of imperfect-information extensive form games include poker and Stratego.

In perfect information extensive-form games, algorithms based on minimax tree search have had success on games such as checkers, chess and Go [Silver et al., 2017]. Extensive-form fictitious play (XFP) [Heinrich et al., 2015] and counterfactual regret minimization (CFR) [Zinkevich et al., 2008] extend fictitious play and regret matching, respectively, to extensive form games. In large imperfect information games such as heads up no-limit Texas Hold 'em, counterfactual regret minimization has been used on an abstracted version of the game to beat top humans [Brown and Sandholm, 2018]. However, this is not a general method because finding abstractions requires expert domain knowledge and cannot be easily done for different games. For very large imperfect information games such as Barrage Stratego, it is not clear how to use abstractions and CFR. Deep CFR [Brown et al., 2019] is a general method that trains a neural network on a buffer of counterfactual values. However, Deep CFR uses external sampling, which may be impractical for games with a large branching factor such as Stratego and Barrage Stratego. DREAM [Steinberger et al., 2020] and ARMAC [Gruslys et al., 2020] are model-free regret-based deep learning approaches. Current Barrage Stratego bots are based on imperfect information tree search and are unable to beat even intermediate-level human players [Schadd and Winands, 2009, Jug and Schadd, 2009].

Recently, deep reinforcement learning has proven effective on high-dimensional sequential decision making problems such as Atari games and robotics [Li, 2017]. AlphaStar [Vinyals et al., 2019] beat top humans at Starcraft using self-play and population-based reinforcement learning. Similarly, OpenAI Five [Berner et al., 2019] beat top humans at Dota using self play reinforcement learning. Similar population-based methods have achieved human-level performance on Capture the Flag [Jaderberg et al., 2019]. However, these algorithms are not guaranteed to converge to an approximate Nash equilibrium. Neural Fictitious Self Play (NFSP) [Heinrich and Silver, 2016] approximates extensive-form fictitious play by progressively training a best response against an average of all past policies using reinforcement learning. The average policy is represented by a neural network and is trained via supervised learning using a replay buffer of past best response actions. This replay buffer may become prohibitively large in complex games.

2.1 Policy Space Response Oracles

The Double Oracle algorithm [McMahan et al., 2003] is an algorithm for finding a Nash equilibrium in normal form games. The algorithm works by keeping a population of policies $\Pi^t \subset \Pi$ at time t. Each iteration a Nash equilibrium $\sigma^{*,t}$ is computed for the game restricted to policies in Π^t. Then, a best response to this Nash equilibrium for each player $\text{BR}(\sigma^{*,t}_{-i})$ is computed and added to the population $\Pi^{t+1}_i = \Pi^t_i \cup \{\text{BR}(\sigma^{*,t}_{-i})\}$ for $i \in \{1, 2\}$.

Policy Space Response Oracles (PSRO) approximates the Double Oracle algorithm. The meta Nash equilibrium is computed on the empirical game matrix U^{Π}, given by having each policy in the population Π play each other policy and tracking average utility in a payoff matrix. In each iteration, an approximate best response to the current meta Nash equilibrium over the policies is computed via any reinforcement learning algorithm. In this work we use a discrete-action version of Soft Actor Critic (SAC), described in Section 3.1.

One issue with PSRO is that it is based on a normal-form algorithm, and the number of pure strategies in a normal form representation of an extensive-form game is exponential in the number of information sets. In practice, however, PSRO is able to achieve good performance in large games,

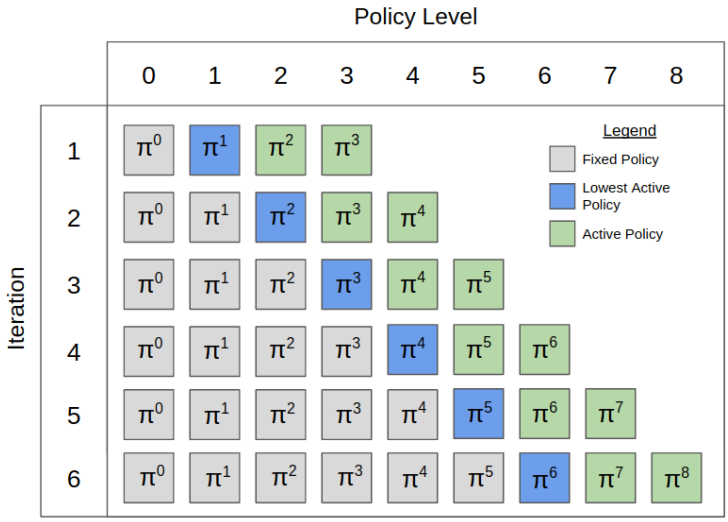


Figure 1: Pipeline PSRO. The lowest-level active policy π^j (blue) plays against the meta Nash equilibrium $\sigma^{*,j}$ of the lower-level fixed policies in Π^f (gray). Each additional active policy (green) plays against the meta Nash equilibrium of the fixed and training policies in levels below it. Once the lowest active policy plateaus, it becomes fixed, a new active policy is added, and the next active policy becomes the lowest active policy. In the first iteration, the fixed population consists of a single random policy.

possibly because large sections of the game tree correspond to weak actions, so only a subset of pure strategies need be enumerated for satisfactory performance. Another issue with PSRO is that it is a sequential algorithm, requiring a full best response computation in every iteration. This paper addresses the latter problem by parallelizing PSRO while maintaining the same convergence guarantees.

DCH [Lanctot et al., 2017] parallelizes PSRO by training multiple reinforcement learning agents, each against the meta Nash equilibrium of agents below it in the hierarchy. A problem with DCH is that one needs to set the number of workers equal to the number of policies in the final population beforehand. For large games such as Barrage Stratego, this might require hundreds of parallel workers. Also, in practice, DCH fails to converge in small random normal form games even with an exact best-response oracle and a learning rate of 1, because early levels may change their best response occasionally due to randomness in estimation of the meta Nash equilibrium. In our experiments and in the DCH experiments in Lanctot et al. [2017], DCH is unable to achieve low exploitability on Leduc poker.

Another existing parallel PSRO algorithm is Rectified PSRO [Balduzzi et al., 2019]. Rectified PSRO assigns each learner to play against the policies that it currently beats. However, we prove that Rectified PSRO does not converge to a Nash equilibrium in all symmetric zero-sum games. In our experiments, Rectified PSRO rarely converges to an approximate Nash equilibrium in random normal form games.

3 Pipeline Policy Space Response Oracles (P2SRO)

Pipeline PSRO (P2SRO; Algorithm 1) is able to scale up PSRO with convergence guarantees by maintaining a hierarchical pipeline of reinforcement learning policies, each training against the policies in the lower levels of the hierarchy (Figure 1). P2SRO has two classes of policies: fixed and active. The set of fixed policies are denoted by Π^f and do not train anymore, but remain in the fixed population. The parallel reinforcement learning workers train the active policies, denoted Π^a in a hierarchical pipeline, training against the meta Nash equilibrium distribution of both the fixed policies and the active policies in levels below them in the pipeline. The entire population Π consists of the union of Π^f and Π^a. For each policy π_i^j in the active policies Π_i^a, to compute the distribution

Algorithm 1 Pipeline Policy-Space Response Oracles

Input: Initial policy sets for all players Π^f
Compute expected utilities for empirical payoff matrix U^{Π} for each joint $\pi \in \Pi$
Compute meta-Nash equilibrium $\sigma^{*,j}$ over fixed policies (Π^f)
for many episodes **do**
  for all $\pi^j \in \Pi^a$ in parallel **do**
    for player $i \in \{1, 2\}$ **do**
      Sample $\pi_{-i} \sim \sigma^{*,j}_{-i}$
      Train π^j_i against π_{-i}
    end for
    if π^j plateaus and π^j is the lowest active policy **then**
      $\Pi^f = \Pi^f \cup \{\pi^j\}$
      Initialize new active policy at a higher level than all existing active policies
      Compute missing entries in U^{Π} from Π
      Compute meta Nash equilibrium for each active policy
    end if
    Periodically compute meta Nash equilibrium for each active policy
  end for
end for
Output current meta Nash equilibrium on whole population σ^*

of policies to train against, a meta Nash equilibrium $\sigma^{*,j}_{-i}$ is periodically computed on policies lower than π^j_i: $\Pi^f_{-i} \cup \{\pi^k_{-i} \in \Pi^a_{-i} | k < j\}$ and π^j_i trains against this distribution.

The performance of a policy π^j is given by the average performance during training $\mathbb{E}_{\pi_1 \sim \sigma^{*,j}_1}[u_2(\pi_1, \pi^j_2)] + \mathbb{E}_{\pi_2 \sim \sigma^{*,j}_2}[u_1(\pi^j_1, \pi_2)]$ against the meta Nash equilibrium distribution $\sigma^{*,j}$. Once the performance of the lowest-level active policy π^j in the pipeline does not improve past a given threshold in a given amount of time, we say that the policy's performance plateaus, and π^j becomes fixed and is added to the fixed population Π^f. Once π^j is added to the fixed population Π^f, then π^{j+1} becomes the new lowest active policy. A new policy is initialized and added as the highest-level policy in the active policies Π^a. Because the lowest-level policy only trains against the previous fixed policies Π^f, P2SRO maintains the same convergence guarantees as PSRO. Unlike PSRO, however, each policy in the pipeline above the lowest-level policy is able to get a head start by pre-training against the moving target of the meta Nash equilibrium of the policies below it. Unlike Rectified PSRO and DCH, P2SRO converges to an approximate Nash equilibrium across a variety of imperfect information games such as Leduc Poker and random normal form games.

In our experiments we model the non-symmetric games of Leduc poker and Barrage Stratego as symmetric games by training one policy that can observe which player it is at the start of the game and play as either the first or the second player. We find that in practice it is more efficient to only train one population than to train two different populations, especially in larger games, such as Barrage Stratego.

3.1 Implementation Details

For the meta Nash equilibrium solver we use fictitious play [Fudenberg et al., 1998]. Fictitious play is a simple method for finding an approximate Nash equilibrium in normal form games. Every iteration, a best response to the average strategy of the population is added to the population. The average strategy converges to an approximate Nash equilibrium. For the approximate best response oracle, we use a discrete version of Soft Actor Critic (SAC) [Haarnoja et al., 2018, Christodoulou, 2019]. We modify the version used in RLlib [Liang et al., 2018, Moritz et al., 2018] to account for discrete actions.

3.2 Analysis

PSRO is guaranteed to converge to an approximate Nash equilibrium and doesn't need a large replay buffer, unlike NFSP and Deep CFR. In the worst case, all policies in the original game must be added

before PSRO reaches an approximate Nash equilibrium. Empirically, on random normal form games, PSRO performs better than selecting pure strategies at random without replacement. This implies that in each iteration, PSRO is more likely than random to add a pure strategy that is part of the support of the Nash equilibrium of the full game, suggesting the conjecture that PSRO has faster convergence rate than random strategy selection. The following theorem indirectly supports this conjecture.

Theorem 3.1. *Let σ be a Nash equilibrium of a symmetric normal form game (Π, U) and let Π^e be the set of pure strategies in its support. Let $\Pi' \subset \Pi$ be a population that does not cover $\Pi^e \not\subseteq \Pi'$, and let σ' be the meta Nash equilibrium of the original game restricted to strategies in Π'. Then there exists a pure strategy $\pi \in \Pi^e \setminus \Pi'$ such that π does not lose to σ'.*

Proof. Contained in supplementary material. □

Ideally, PSRO would be able to add a member of $\Pi^e \setminus \Pi'$ to the current population Π' at each iteration. However, the best response to the current meta Nash equilibrium σ' is generally not a member of Π^e. Theorem 3.1 shows that for an *approximate* best response algorithm with a weaker guarantee of not losing to σ', it is possible that a member of $\Pi^e \setminus \Pi'$ is added at each iteration.

Even assuming that a policy in the Nash equilibrium support is added at each iteration, the convergence of PSRO to an approximate Nash equilibrium can be slow because each policy is trained sequentially by a reinforcement learning algorithm. DCH, Rectified PSRO, and P2SRO are methods of speeding up PSRO through parallelization. In large games, many of the basic skills (such as extracting features from the board) may need to be relearned when starting each iteration from scratch. DCH and P2SRO are able to speed up PSRO by pre-training each level on the moving target of the meta Nash equilibrium of lower-level policies before those policies converge. This speedup would be linear with the number of parallel workers if each policy could train on the fixed final meta Nash equilibrium of the policies below it. Since it trains instead on a moving target, we expect the speedup to be sub-linear in the number of workers.

DCH is an approximation of PSRO that is not guaranteed to converge to an approximate Nash equilibrium if the number of levels is not equal to the number of pure strategies in the game, and is in fact guaranteed *not* to converge to an approximate Nash equilibrium if the number of levels cannot support it.

Another parallel PSRO algorithm, Rectified PSRO, is not guaranteed to converge to an approximate Nash equilibrium.

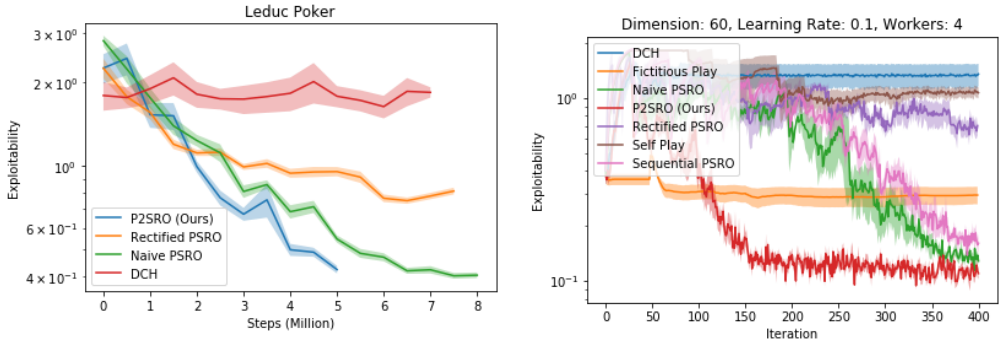
Proposition 3.1. *Rectified PSRO with an oracle best response does not converge to a Nash equilibrium in all symmetric two-player, zero-sum normal form games.*

Proof. Consider the following symmetric two-player zero-sum normal form game:

$$\begin{bmatrix} 0 & -1 & 1 & -\frac{2}{5} \\ 1 & 0 & -1 & -\frac{2}{5} \\ -1 & 1 & 0 & -\frac{2}{5} \\ \frac{2}{5} & \frac{2}{5} & \frac{2}{5} & 0 \end{bmatrix}$$

This game is based on Rock–Paper–Scissors, with an extra strategy added that beats all other strategies and is the pure Nash equilibrium of the game. Suppose the population of Rectified PSRO starts as the pure Rock strategy.

- Iteration 1: Rock ties with itself, so a best response to Rock (Paper) is added to the population.
- Iteration 2: The meta Nash equilibrium over Rock and Paper has all mass on Paper. The new strategy that gets added is the best response to Paper (Scissors).
- Iteration 3: The meta Nash equilibrium over Rock, Paper, and Scissors equally weights each of them. Now, for each of the three strategies, Rectified PSRO adds a best response to the meta-Nash-weighted combination of strategies that it beats or ties. Since Rock beats or ties Rock and Scissors, a best response to a $50 - 50$ combination of Rock and Scissors is Rock,



(a) Leduc poker

(b) Random Symmetric Normal Form Games

Figure 2: Exploitability of Algorithms on Leduc poker and Random Symmetric Normal Form Games

with an expected utility of $\frac{1}{2}$. Similarly, for Paper, since Paper beats or ties Paper and Rock, a best response to a $50 - 50$ combination of Paper and Rock is Paper. For Scissors, the best response for an equal mix of Scissors and Paper is Scissors. So in this iteration no strategy is added to the population and the algorithm terminates.

We see that the algorithm terminates without expanding the fourth strategy. The meta Nash equilibrium of the first three strategies that Rectified PSRO finds are not a Nash equilibrium of the full game, and are exploited by the fourth strategy, which is guaranteed to get a utility of $\frac{2}{5}$ against any mixture of them. □

The pattern of the counterexample presented here is possible to occur in large games, which suggests that Rectified PSRO may not be an effective algorithm for finding an approximate Nash equilibrium in large games. Prior work has found that Rectified PSRO does not converge to an approximate Nash equilibrium in Kuhn Poker [Muller et al., 2020].

Proposition 3.2. *P2SRO with an oracle best response converges to a Nash equilibrium in all two-player, zero-sum normal form games.*

Proof. Since only the lowest active policy can be submitted to the fixed policies, this policy is an oracle best response to the meta Nash distribution of the fixed policies, making P2SRO with an oracle best response equivalent to the Double Oracle algorithm. □

Unlike DCH which becomes unstable when early levels change, P2SRO is able to avoid this problem because early levels become fixed once they plateau. While DCH only approximates PSRO, P2SRO has equivalent guarantees to PSRO because the lowest active policy always trains against a fixed meta Nash equilibrium before plateauing and becoming fixed itself. This fixed meta Nash distribution that it trains against is in principle the same as the one that PSRO would train against. The only difference between P2SRO and PSRO is that the extra workers in P2SRO are able to get a head-start by pre-training on lower level policies while those are still training. Therefore, P2SRO inherits the convergence guarantees from PSRO while scaling up when multiple processors are available.

4 Results

We compare P2SRO with DCH, Rectified PSRO, and a naive way of parallelizing PSRO that we term Naive PSRO. Naive PSRO is a way of parallelizing PSRO where each additional worker trains against the same meta Nash equilibrium of the fixed policies. Naive PSRO is beneficial when randomness in the reinforcement learning algorithm leads to a diversity of trained policies, and in our experiments it performs only slightly better than PSRO. Additionally, in random normal form game experiments, we include the original, non-parallel PSRO algorithm, termed sequential PSRO, and non-parallelized self-play, where a single policy trains against the latest policy in the population.

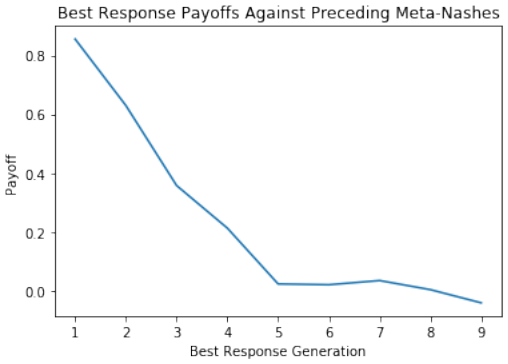


Figure 3: Barrage Best Response Payoffs Over Time

Name	P2SRO Win Rate vs. Bot
Asmodeus	81%
Celsius	70%
Vixen	69%
Celsius1.1	65%
All Bots Average	**71%**

Table 1: Barrage P2SRO Results vs. Existing Bots

We find that DCH fails to reliably converge to an approximate Nash equilibrium across random symmetric normal form games and small poker games. We believe this is because early levels can randomly change even after they have plateaued, causing instability in higher levels. In our experiments, we analyze the behavior of DCH with a learning rate of 1 in random normal form games. We hypothesized that DCH with a learning rate of 1 would be equivalent to the double oracle algorithm and converge to an approximate Nash. However, we found that the best response to a fixed set of lower levels can be different in each iteration due to randomness in calculating a meta Nash equilibrium. This causes a ripple effect of instability through the higher levels. We find that DCH almost never converges to an approximate Nash equilibrium in random normal form games.

Although not introduced in the original paper, we find that DCH converges to an approximate Nash equilibrium with an annealed learning rate. An annealed learning rate allows early levels to not continually change, so the variance of all of the levels can tend to zero. Reinforcement learning algorithms have been found to empirically converge to approximate Nash equilibria with annealed learning rates [Srinivasan et al., 2018, Bowling and Veloso, 2002]. We find that DCH with an annealed learning rate does converge to an approximate Nash equilibrium, but it can converge slowly depending on the rate of annealing. Furthermore, annealing the learning rate can be difficult to tune with deep reinforcement learning, and can slow down training considerably.

4.1 Random Symmetric Normal Form Games

For each experiment, we generate a random symmetric zero-sum normal form game of dimension n by generating a random antisymmetric matrix P. Each element in the upper triangle is distributed uniformly: $\forall i < j \leq n, a_{i,j} \sim \text{UNIFORM}(-1, 1)$. Every element in the lower triangle is set to be the negative of its diagonal counterpart: $\forall j < i \leq n, a_{i,j} = -a_{j,i}$. The diagonal elements are equal to zero: $a_{i,i} = 0$. The matrix defines the utility of two pure strategies to the row player. A strategy $\pi \in \Delta^n$ is a distribution over the n pure strategies of the game given by the rows (or equivalently, columns) of the matrix. In these experiments we can easily compute an exact best response to a strategy and do not use reinforcement learning to update each strategy. Instead, as a strategy π "trains" against another strategy $\hat{\pi}$, it is updated by a learning rate r multiplied by the best response to that strategy: $\pi' = r\text{BR}(\hat{\pi}) + (1 - r)\pi$.

Figure 2 show results for each algorithm on random symmetric normal form games of dimension 60, about the same dimension of the normal form of Kuhn poker. We run each algorithm on five different random symmetric normal form games. We report the mean exploitability over time of these algorithms and add error bars corresponding to the standard error of the mean. P2SRO reaches an approximate Nash equilibrium much faster than the other algorithms. Additional experiments on different dimension games and different learning rates are included in the supplementary material. In each experiment, P2SRO converges to an approximate Nash equilibrium much faster than the other algorithms.

4.2 Leduc Poker

Leduc poker is played with a deck of six cards of two suits with three cards each. Each player bets one chip as an ante, then each player is dealt one card. After, there is a a betting round and then another card is dealt face up, followed by a second betting round. If a player's card is the same rank as the public card, they win. Otherwise, the player whose card has the higher rank wins. We run the following parallel PSRO algorithms on Leduc: P2SRO, DCH, Rectified PSRO, and Naive PSRO. We run each algorithm for three random seeds with three workers each. Results are shown in Figure 2. We find that P2SRO is much faster than the other algorithms, reaching 0.4 exploitability almost twice as soon as Naive PSRO. DCH and Rectified PSRO never reach a low exploitability.

4.3 Barrage Stratego

Barrage Stratego is a smaller variant of the board game Stratego that is played competitively by humans. The board consists of a ten-by-ten grid with two two-by-two barriers in the middle. Initially, each player only knows the identity of their own eight pieces. At the beginning of the game, each player is allowed to place these pieces anywhere on the first four rows closest to them. More details about the game are included in the supplementary material.

We find that the approximate exploitability of the meta-Nash equilibrium of the population decreases over time as measured by the performance of each new best response. This is shown in Figure 3, where the payoff is 1 for winning and -1 for losing. We compare to all existing bots that are able to play Barrage Stratego. These bots include: Vixen, Asmodeus, and Celsius. Other bots such as Probe and Master of the Flag exist, but can only play Stratego and not Barrage Stratego. We show results of P2SRO against the bots in Table 1. We find that P2SRO is able to beat these existing bots by 71% on average after $820,000$ episodes, and has a win rate of over 65% against each bot. We introduce an open-source environment for Stratego, Barrage Stratego, and smaller Stratego games at `https://github.com/JBLanier/stratego_env`.

Broader Impact

Stratego and Barrage Stratego are very large imperfect information board games played by many around the world. Although variants of self-play reinforcement learning have achieved grandmaster level performance on video games, it is unclear if these algorithms could work on Barrage Stratego or Stratego because they are not principled and fail on smaller games. We believe that P2SRO will be able to achieve increasingly good performance on Barrage Stratego and Stratego as more time and compute are added to the algorithm. We are currently training P2SRO on Barrage Stratego and we hope that the research community will also take interest in beating top humans at these games as a challenge and inspiration for artificial intelligence research.

This research focuses on how to scale up algorithms for computing approximate Nash equilibria in large games. These methods are very compute-intensive when applied to large games. Naturally, this favors large tech companies or governments with enough resources to apply this method for large, complex domains, including in real-life scenarios such as stock trading and e-commerce. It is hard to predict who might be put at an advantage or disadvantage as a result of this research, and it could be argued that powerful entities would gain by reducing their exploitability. However, the same players already do and will continue to benefit from information and computation gaps by exploiting suboptimal behavior of disadvantaged parties. It is our belief that, in the long run, preventing exploitability and striving as much as practical towards a provably efficient equilibrium can serve to level the field, protect the disadvantaged, and promote equity and fairness.

Acknowledgments and Disclosure of Funding

SM and PB in part supported by grant NSF 1839429 to PB.

References

D. Balduzzi, M. Garnelo, Y. Bachrach, W. Czarnecki, J. Perolat, M. Jaderberg, and T. Graepel. Open-ended learning in symmetric zero-sum games. In *International Conference on Machine*

Learning, pages 434–443, 2019.

C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.

N. Brown and T. Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

N. Brown, A. Lerer, S. Gross, and T. Sandholm. Deep counterfactual regret minimization. In *International Conference on Machine Learning*, pages 793–802, 2019.

P. Christodoulou. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

D. Fudenberg, F. Drew, D. K. Levine, and D. K. Levine. *The theory of learning in games*. The MIT Press, 1998.

A. Gruslys, M. Lanctot, R. Munos, F. Timbers, M. Schmid, J. Perolat, D. Morrill, V. Zambaldi, J.-B. Lespiau, J. Schultz, et al. The advantage regret-matching actor-critic. *arXiv preprint arXiv:2008.12234*, 2020.

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870, 2018.

J. Heinrich and D. Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016.

J. Heinrich, M. Lanctot, and D. Silver. Fictitious self-play in extensive-form games. In *International Conference on Machine Learning*, pages 805–813, 2015.

M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castaneda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

S. Jug and M. Schadd. The 3rd stratego computer world championship. *Icga Journal*, 32(4):233, 2009.

M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Pérolat, D. Silver, and T. Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4190–4203, 2017.

Y. Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pages 3053–3062, 2018.

H. B. McMahan, G. J. Gordon, and A. Blum. Planning in the presence of cost functions controlled by an adversary. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 536–543, 2003.

P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 561–577, 2018.

P. Muller, S. Omidshafiei, M. Rowland, K. Tuyls, J. Perolat, S. Liu, D. Hennes, L. Marris, M. Lanctot, E. Hughes, et al. A generalized training approach for multiagent learning. *International Conference on Learning Representations (ICLR)*, 2020.

N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.

M. Schadd and M. Winands. Quiescence search for stratego. In *Proceedings of the 21st Benelux Conference on Artificial Intelligence. Eindhoven, the Netherlands*, 2009.

Y. Shoham and K. Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. v. d. Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 10 2017. ISSN 0028-0836. doi: 10.1038/nature24270. URL `http:https://doi.org/10.1038/nature24270`.

S. Srinivasan, M. Lanctot, V. Zambaldi, J. Pérolat, K. Tuyls, R. Munos, and M. Bowling. Actor-critic policy optimization in partially observable multiagent environments. In *Advances in neural information processing systems*, pages 3422–3435, 2018.

E. Steinberger, A. Lerer, and N. Brown. Dream: Deep regret minimization with advantage baselines and model-free learning. *arXiv preprint arXiv:2006.10410*, 2020.

P. D. Taylor and L. B. Jonker. Evolutionary stable strategies and game dynamics. *Mathematical biosciences*, 40(1-2):145–156, 1978.

O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret minimization in games with incomplete information. In *Advances in neural information processing systems*, pages 1729–1736, 2008.

A Proofs of Theorems

Theorem A.1. *Let σ be a Nash equilibrium of a symmetric normal form game (Π, U) and let Π^e be the set of pure strategies in its support. Let $\Pi' \subset \Pi$ be a population that does not cover $\Pi^e \not\subseteq \Pi'$, and let σ' be the meta Nash equilibrium of the original game restricted to strategies in Π'. Then there exists a pure strategy $\pi \in \Pi^e \setminus \Pi'$ such that π does not lose to σ'.*

Proof. σ' is a meta Nash equilibrium, implying $\sigma'^\intercal G \sigma' = 0$, where G is the payoff matrix for the row player. In fact, each policy π in the support Π'^e of σ' has $1_\pi^\intercal G \sigma' = 0$, where 1_π is the one-hot encoding of π in Π.

Consider the sets $\Pi^+ = \{\pi : \sigma(\pi) > \sigma'(\pi)\} = \Pi^e \setminus \Pi'$ and $\Pi^- = \{\pi : \sigma(\pi) < \sigma'(\pi)\} \subseteq \Pi'^e$. Note the assumption that Π^+ is not empty. If each $\pi \in \Pi^+$ had $1_\pi^\intercal G \sigma' < 0$, we would have

$$\begin{aligned}\sigma^\intercal G \sigma' = (\sigma - \sigma')^\intercal G \sigma' &= \sum_{\pi \in \Pi^+} (\sigma(\pi) - \sigma'(\pi)) 1_\pi^\intercal G \sigma' + \sum_{\pi \in \Pi^-} (\sigma(\pi) - \sigma'(\pi)) 1_\pi^\intercal G \sigma' \\ &= \sum_{\pi \in \Pi^+} (\sigma(\pi) - \sigma'(\pi)) 1_\pi^\intercal G \sigma' < 0,\end{aligned}$$

in contradiction to σ being a Nash equilibrium. We conclude that there must exist $\pi \in \Pi^+$ with $1_\pi^\intercal G \sigma' \geq 0$. □

B Barrage Stratego Details

Barrage is a smaller variant of the board game Stratego that is played competitively by humans. The board consists of a ten-by-ten grid with two two-by-two barriers in the middle (see image for details). Each player has eight pieces, consisting of one Marshal, one General, one Miner, two Scouts, one

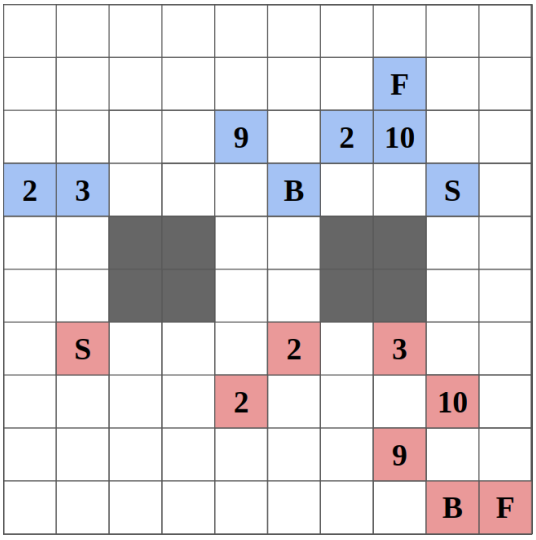


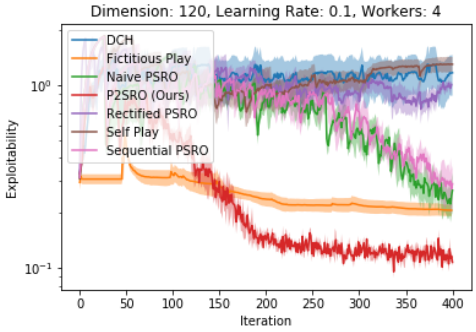
Figure 4: Valid Barrage Stratego Setup (note that the piece values are not visible to the other player)

Spy, one Bomb, and one Flag. Crucially, each player only knows the identity of their own pieces. At the beginning of the game, each player is allowed to place these pieces anywhere on the first four rows closest to them.

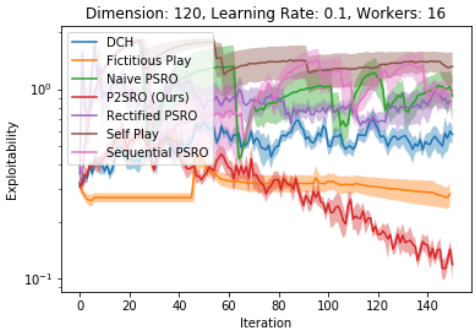
The Marshal, General, Spy, and Miner may move only one step to any adjacent space but not diagonally. Bomb and Flag pieces cannot be moved. The Scout may move in a straight line like a rook in chess. A player can attack by moving a piece onto a square occupied by an opposing piece. Both players then reveal their piece's rank and the weaker piece gets removed. If the pieces are of equal rank then both get removed. The Marshal has higher rank than all other pieces, the General has higher rank than all other beside the Marshal, the Miner has higher rank than the Scout, Spy, Flag, and Bomb, the Scout has higher rank than the Spy and Flag, and the Spy has higher rank than the Flag and the Marshal when it attacks the Marshal. Bombs cannot attack but when another piece besides the Miner attacks a Bomb, the Bomb has higher rank. The player who captures his/her opponent's Flag or prevents the other player from moving any piece wins.

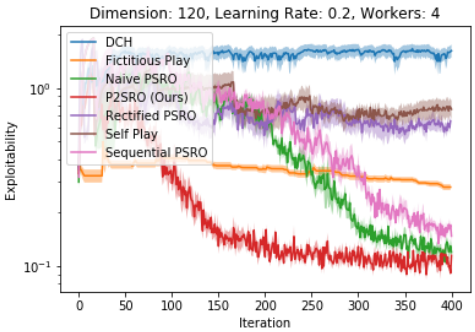
C Additional Random Normal Form Games Results

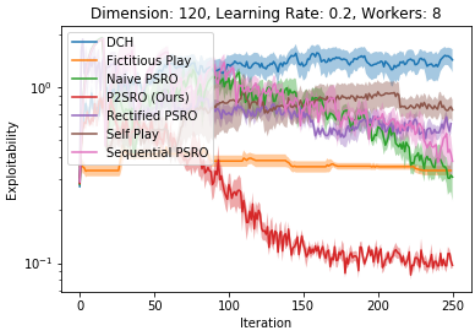
We compare the exploitability over time of P2SRO with DCH, Naive PSRO, Rectified PSRO, Self Play, and Sequential PSRO. We run 5 experiments for each set of dimension, learning rate, and number of parallel workers and record the average exploitability over time. We run experiments on dimensions of size 15, 30, 45, 60, and 120, learning rates of 0.1, 0.2, and 0.5, and 4, 8, and 16 parallel workers. We find that not only does P2SRO converge to an approximate Nash equilibrium in every experiment, but that it performs as good as or better than all other algorithms in every experiment. We also find that the relative performance of P2SRO versus the other algorithms seems to improve as the dimension of the game improves.

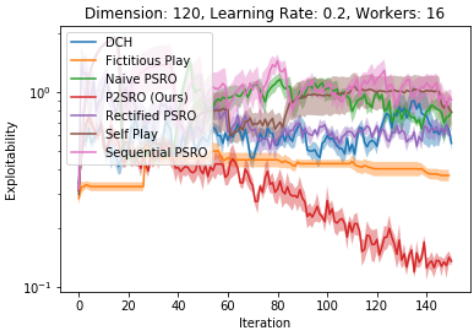

Dimension: 120, Learning Rate: 0.1, Workers: 4
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

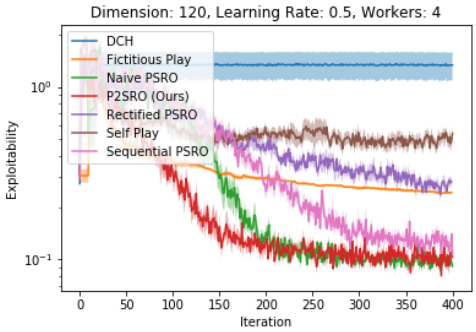

Dimension: 120, Learning Rate: 0.1, Workers: 8
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration


Dimension: 120, Learning Rate: 0.1, Workers: 16
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

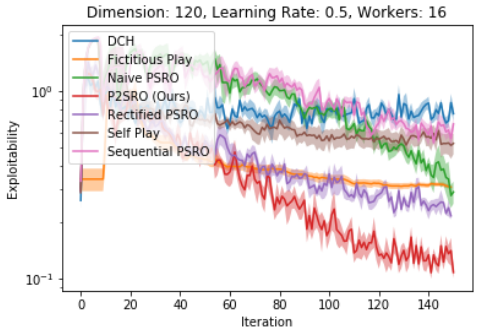

Dimension: 120, Learning Rate: 0.2, Workers: 4
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

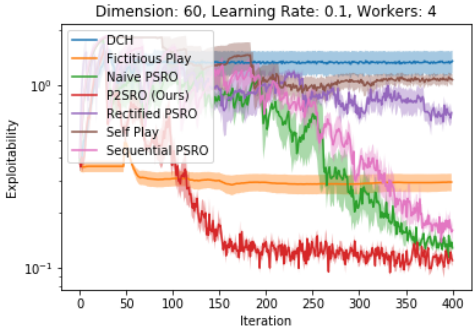

Dimension: 120, Learning Rate: 0.2, Workers: 8
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

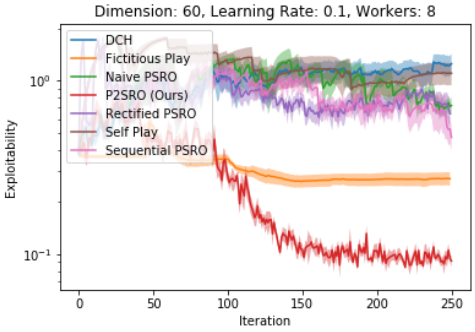

Dimension: 120, Learning Rate: 0.2, Workers: 16
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

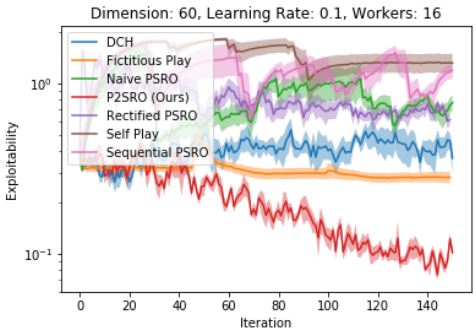

Dimension: 120, Learning Rate: 0.5, Workers: 4
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

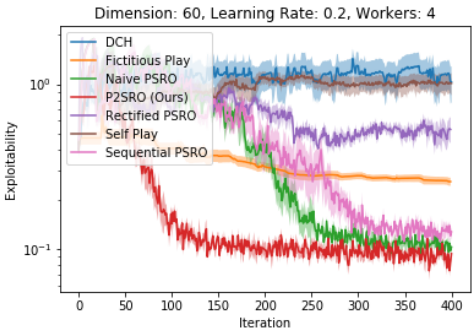

Dimension: 120, Learning Rate: 0.5, Workers: 8
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

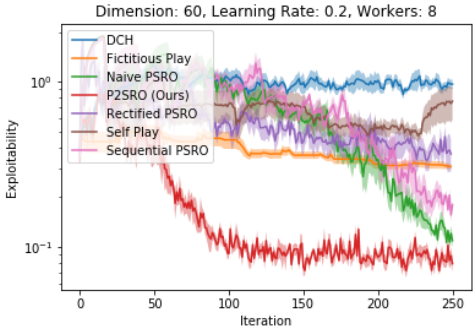

Dimension: 120, Learning Rate: 0.5, Workers: 16
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

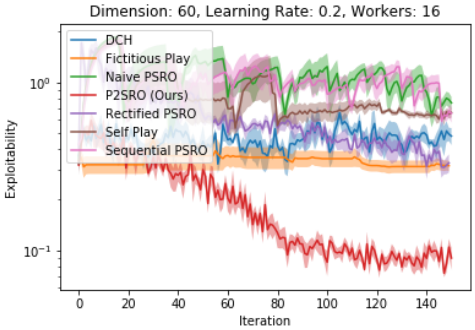

Dimension: 60, Learning Rate: 0.1, Workers: 4
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

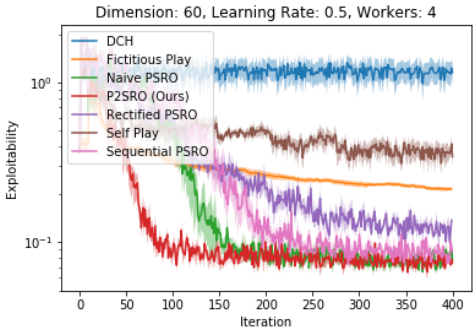

Dimension: 60, Learning Rate: 0.1, Workers: 8
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

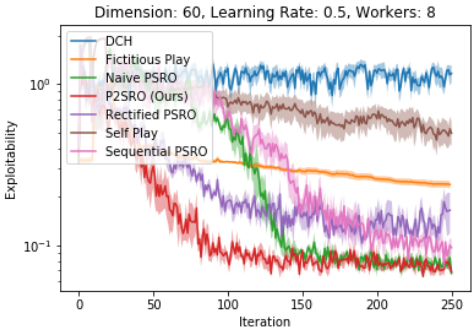

Dimension: 60, Learning Rate: 0.1, Workers: 16
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

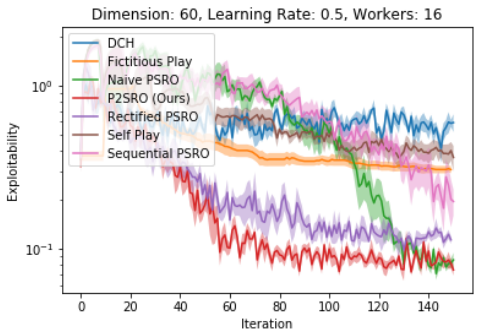

Dimension: 60, Learning Rate: 0.2, Workers: 4
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

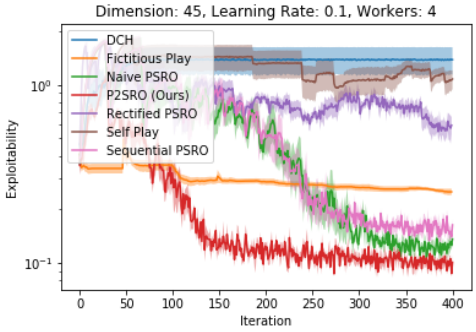

Dimension: 60, Learning Rate: 0.2, Workers: 8
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

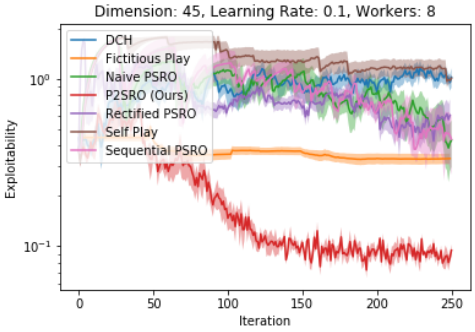

Dimension: 60, Learning Rate: 0.2, Workers: 16
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

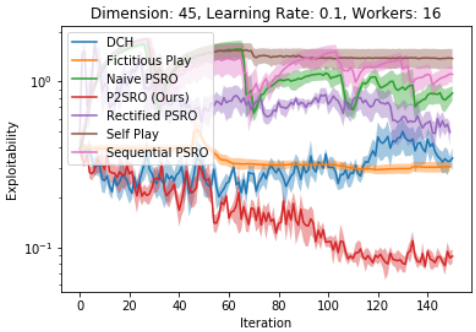

Dimension: 60, Learning Rate: 0.5, Workers: 4
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

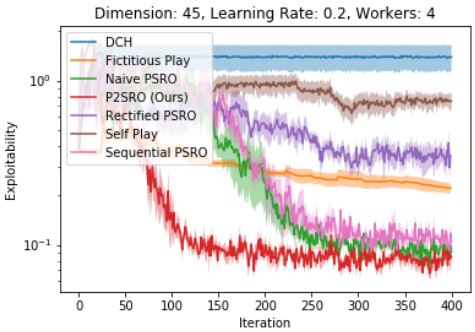

Dimension: 60, Learning Rate: 0.5, Workers: 8
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

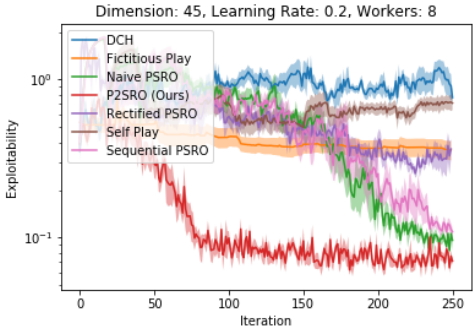

Dimension: 60, Learning Rate: 0.5, Workers: 16
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration


Dimension: 45, Learning Rate: 0.1, Workers: 4
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

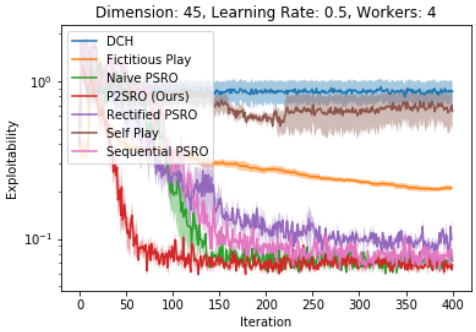

Dimension: 45, Learning Rate: 0.1, Workers: 8
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

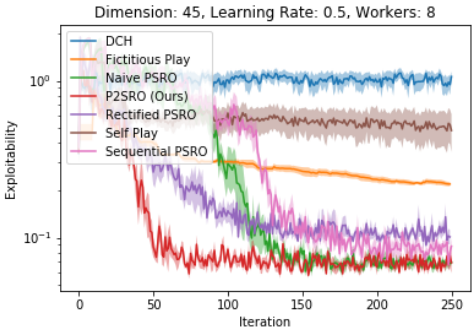

Dimension: 45, Learning Rate: 0.1, Workers: 16
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

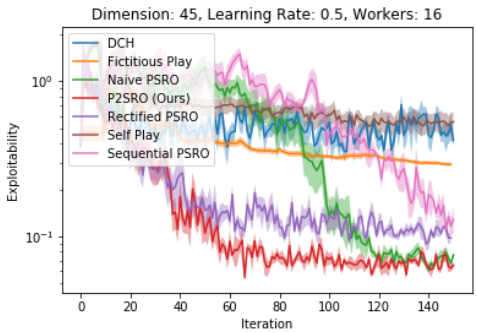

Dimension: 45, Learning Rate: 0.2, Workers: 4
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

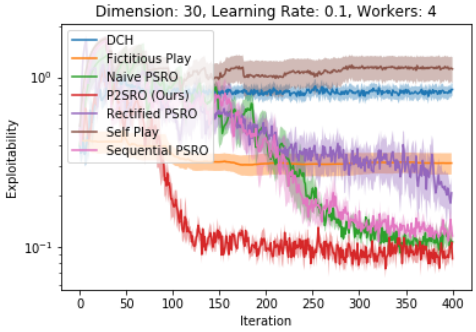

Dimension: 45, Learning Rate: 0.2, Workers: 8
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

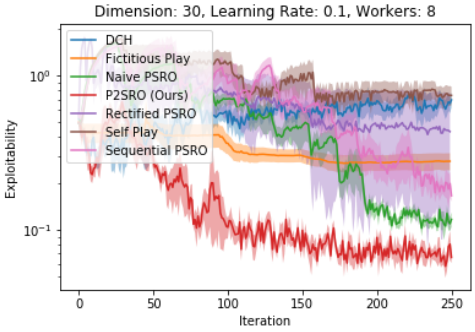

Dimension: 45, Learning Rate: 0.2, Workers: 16
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

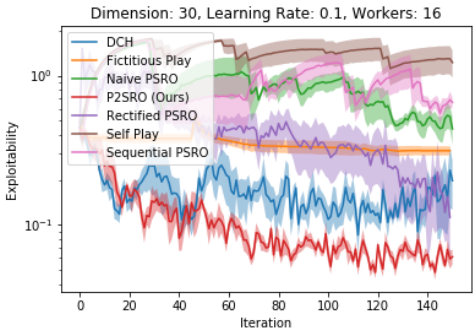

Dimension: 45, Learning Rate: 0.5, Workers: 4
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

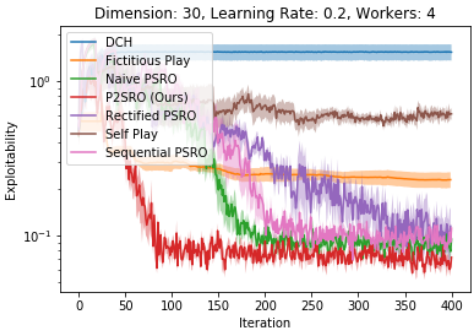

Dimension: 45, Learning Rate: 0.5, Workers: 8
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

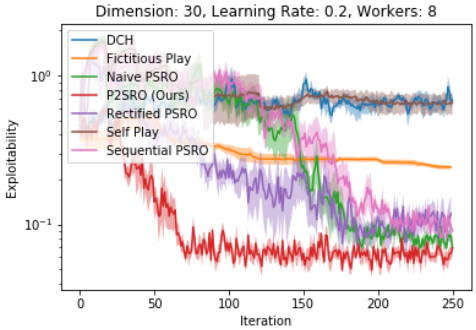

Dimension: 45, Learning Rate: 0.5, Workers: 16
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

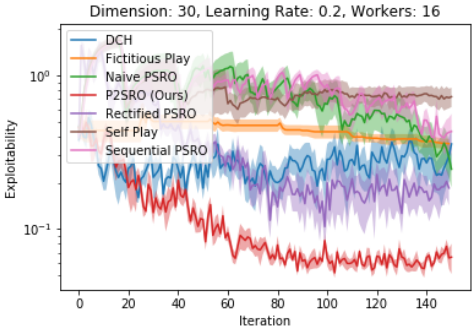

Dimension: 30, Learning Rate: 0.1, Workers: 4
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

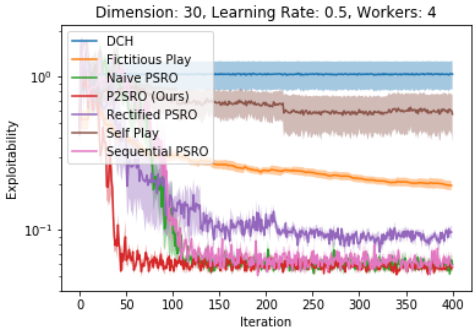

Dimension: 30, Learning Rate: 0.1, Workers: 8
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

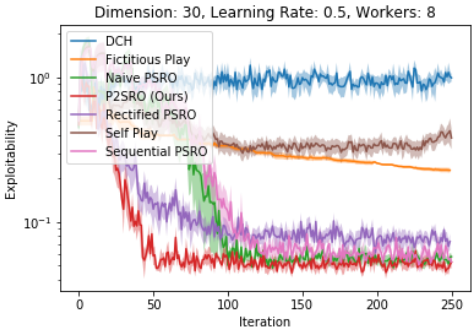

Dimension: 30, Learning Rate: 0.1, Workers: 16
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

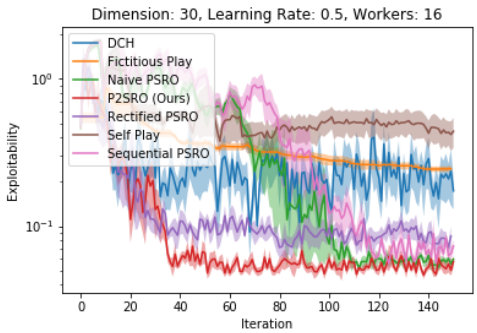

Dimension: 30, Learning Rate: 0.2, Workers: 4
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

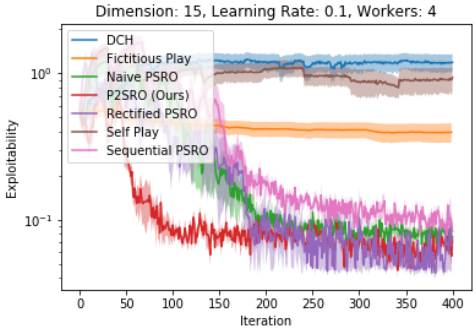

Dimension: 30, Learning Rate: 0.2, Workers: 8
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration


Dimension: 30, Learning Rate: 0.2, Workers: 16
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration


Dimension: 30, Learning Rate: 0.5, Workers: 4
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration


Dimension: 30, Learning Rate: 0.5, Workers: 8
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

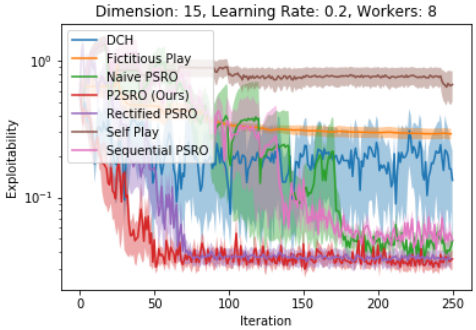

Dimension: 30, Learning Rate: 0.5, Workers: 16
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

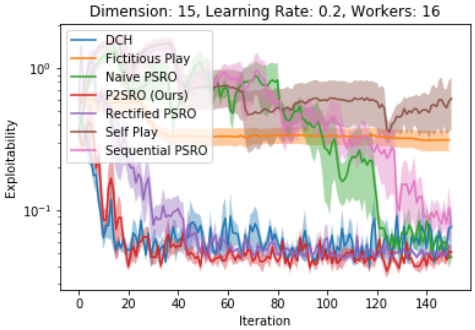

Dimension: 15, Learning Rate: 0.1, Workers: 4
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

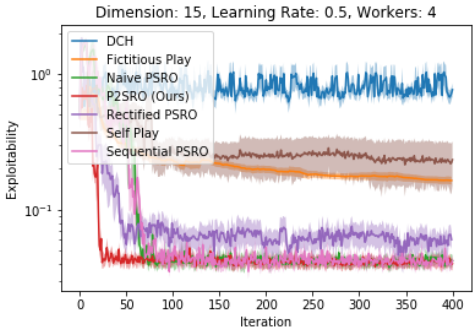

Dimension: 15, Learning Rate: 0.1, Workers: 8
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration

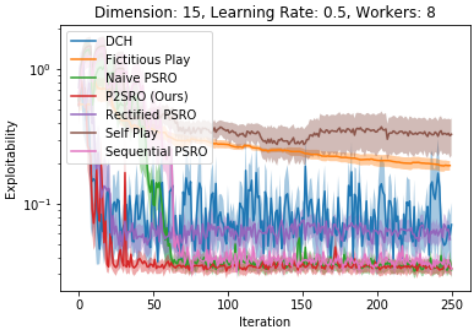

Dimension: 15, Learning Rate: 0.1, Workers: 16
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration


Dimension: 15, Learning Rate: 0.2, Workers: 4
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration


Dimension: 15, Learning Rate: 0.2, Workers: 8
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration


Dimension: 15, Learning Rate: 0.2, Workers: 16
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration


Dimension: 15, Learning Rate: 0.5, Workers: 4
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration


Dimension: 15, Learning Rate: 0.5, Workers: 8
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration


Dimension: 15, Learning Rate: 0.5, Workers: 16
DCH
Fictitious Play
Naive PSRO
P2SRO (Ours)
Rectified PSRO
Self Play
Sequential PSRO
Exploitability
Iteration