Approximated Temporal-Induced Neural Self-Play for Finitely Repeated Bayesian Games

Zihan Zhou,[1] **Zheyuan Ryan Shi,**[2] **Yi Wu,**[3] **Fei Fang**[2]
[1]Shanghai Jiao Tong University, [3]OpenAI, [2]Carnegie Mellon University,
footoredo@sjtu.edu.cn, ryanshi@cmu.edu, jxwuyi@gmail.com, feif@cs.cmu.edu

Abstract

In two-player finitely repeated Bayesian games with one-sided incomplete information, there is a natural information asymmetry among the players. In each round of the game, the player with information disadvantage needs to infer the other player's type from their actions. The other player, knowing that their actions reveal information about themselves, will balance between playing myopically and maintaining information advantage to maximize their accumulated payoff in the long-run, which can lead to deceptive actions. Computing the Perfect Bayesian Nash Equilibrium (PBNE) in such games can be computationally intractable for large games. In this paper, we propose a new learning-based framework to approximate PBNEs, which uses non-parametric approximation and reinforcement learning from self-play. Our initial results show that it can improve the scalability over existing methods and lead to strategy profiles that are close to PBNEs.

1 Introduction

Repeated games with incomplete information is a typical type of games to model long term strategic interactions. In these games, the default solution concept is the Bayesian Nash equilibrium (BNE) (Harsanyi 1968). However, a key problem with BNE is that the equilibrium strategy may not be rational off the equilibrium path. This is problematic, in part, because in the real world everyone is not fully rational and thus may deviate from the equilibrium path. When some player deviates from the equilibrium path, the BNE strategy for the other players might be arbitrarily suboptimal. To fix this problem, a variety of Nash equilibrium refinements have been studied. In this paper, we focus on the perfect Bayesian Nash equilibrium (PBNE) (Cho and Kreps 1987), which is a "first-order refinement" of BNE. PBNE is analogous to the subgame perfect equilibrium in the perfect information games, which ensures that the players' equilibrium strategies are rational at every decision point.

The study of equilibrium refinements is not new in economics (Kreps and Wilson 1982; Bielefeld 1988), and the computational aspect of equilibrium refinements has started to receive some interest recently (An et al. 2011; Etessami et al. 2014; Hansen and Lund 2018). The current effort can be grouped into two classes: mathematical programming-based approaches (Nguyen et al. 2019; Farina and Gatti 2017) and iterative methods (Kroer, Farina, and Sandholm 2017). However, iterative methods only apply to zero-sum games, while mathematical programming-based approaches can hardly scale beyond the simplest toy examples (as detailed in Section 2). On the other hand, deep reinforcement learning (RL) has shown great promise in complex sequential decision-making for both single-agent and multi-agent settings (Mnih et al. 2015; Silver et al. 2018; Kamra et al. 2019). Deep RL often leverages a compact representation of the game's state and the players' action space, which makes it possible to handle large games that are intractable for non-learning-based methods. However, to our knowledge, no prior work has applied reinforcement learning to compute equilibrium refinements in extensive-form games.

As an initial attempt to close the gap between theory and practice, we propose Approximated Temporal-Induced Neural Self-Play, a backward induction-fashioned policy gradient self-play algorithm to solve the PBNE in two-player general-sum finitely repeated Bayesian games. We make the following contributions. (1) *We propose a belief-based learning algorithm to deal with scalability issue*. In finitely repeated Bayesian games, the number of information set in the game tree grows exponentially with the number of time steps. We transform the game into a belief-based game that is easier for learning algorithm to generalize to deal with this problem. (2) *We propose a backward induction fashioned learning schedule to achieve PBNE*. In traditional top-down learning algorithms, the solution quality in a subgame is influenced by the visiting frequency of that state, which may result in unbalanced solution quality among subgames. We eliminate this imbalance by backward induction. (3) *We propose a non-parametric policy approximation algorithm to improve the solution quality*. In our initial attempts, we find neural networks by themselves unable to carry out ideal solutions in our belief-based algorithms due to sample inefficiency. We therefore propose a non-parametric approximation algorithm to alleviate the pressure on neural networks.

2 Related Work

Repeated games with incomplete information have been extensively studied in mathematics and economics (Forges 1992). The literature mostly focuses on infinitely repeated games, where finitely repeated games are used as a tool to analyze the convergence behavior in the former. We develop algorithmic results for finitely repeated games. Generalizing from repeated games to stochastic games (Sorin 2003) poses little technical difficulty to our proposed algorithms. For expository purpose, we use finitely repeated games to illustrate our algorithm. Previous work has proposed algorithms for various models of Bayesian stochastic games with bounded rationality (Chandrasekaran, Chen, and Doshi 2017) or in specific application domains (Albrecht and Ramamoorthy 2013). In contrast, we aim to solve a general repeated Bayesian game with rational agents.

Existing literature on computing refinements of Nash equilibria mainly takes two approaches: the mathematical programming-based approach and the iterative approach. (Nguyen et al. 2019) shares the most similar goal with our paper. They use a high-order mathematical program to compute a PBNE in the Bayesian repeated games. A few works use linear complementary programming to compute other equilibrium refinements such as the extensive-form perfect equilibria (EFPE) (Farina and Gatti 2017) and quasi-perfect equilibria (Miltersen and Sørensen 2010). However, all these mathematical programming based approaches have very limited scalability. On the other hand, iterative approaches like counterfactual regret minimization (Zinkevich et al. 2008) and excessive gap technique (Nesterov 2005) have proven to be scalable for computing NE in large EFGs like poker (Brown and Sandholm 2018). Farina et al. (2017) and Kroer et al. (2017) adapt them to approximate EFPE. However, these approaches are designed specifically for zero-sum games while we consider the significantly harder (PPAD-hard) case of general-sum games.

Deep reinforcement learning can often efficiently solve large scale sequential decision-making problems (Mnih et al. 2015; 2016) in complex games such as Go (Silver et al. 2018). Multi-agent RL deals with the coordination and competition among multiple players and is thus a natural tool for solving EFGs (Hu, Wellman, and others 1998). There has also been real world success like StarCraft (Vinyals et al. 2019). For zero-sum games, various algorithms (Littman 1994; Heinrich, Lanctot, and Silver 2015) are guaranteed to converge to NE in principle. We focus on finding a PBNE, a refinement of NE, in repeated Bayesian games.

3 PBNE in Finitely Repeated Bayesian Games

We consider a *finitely repeated Bayesian game* of two players: player 1 and player 2. Player 2 has a type $\lambda \in \mathbf{\Lambda}$, which is drawn at the beginning of the game according to a public prior distribution $\mathbf{p} = \{p^\lambda : \sum_\lambda p^\lambda = 1, p^\lambda \in [0,1]\}$. While $\mathbf{p}$ is public, the actual type λ is known only to player 2.

The game has T rounds. At round t of the game, both players simultaneously take an action $a_t^1, a_t^2 \in \mathbf{A}$, where $\mathbf{A}$ is a finite action space of size N. Then, player 1 receives a real-valued reward $r^1(a_t^1, a_t^2)$, and player 2 of type λ receives a real-valued reward $r^{2\lambda}(a_t^1, a_t^2)$, based on reward functions $r^1, r^{2\lambda} : A \times A \to \mathbb{R}$. Both players aim to maximize their cumulative reward. A player's action will be visible to the other player at the end of the round.

The history is a tuple of all the actions both players have taken up to a particular round, and thus the set of histories is $H = \bigcup_{t=0}^T H_t = \bigcup_{t=0}^T (A \times A)^t$.

Player 1's strategy x and a type λ player 2's strategy y^λ are mappings from the set of histories $H \setminus H_T$ to a distribution over actions A. We assume perfect recall, and thus it suffices to consider the behavioral strategies. We use $x(i \mid h_t)$ to represent player 1's probability of taking action $i \in A$ in round $t+1$ after a history h_t, and use $y^\lambda(j \mid h_t)$ to represent the probability of the player 2 of type λ taking action $j \in A$ in round $t+1$ after a history h_t.

Although player 1 does not know player 2's type λ, player 1 may form and update belief of the type according to the action player 2 chooses at every round if he knows player 2's strategy. Specifically, given y^λ, $\forall \lambda$, after a history $h_t = (a_1^1, a_1^2, \ldots, a_t^1, a_t^2)$, player 1 uses Bayes rule to update a posterior belief of $p(\lambda | h_t) \propto p^\lambda \prod_{\tau=1}^t y^\lambda(a_\tau^2 \mid h_\tau)$ for each $\lambda \in \Lambda$.

For the last time step T, with a history h_{T-1}, two players' expected utilities are

$$\begin{aligned}
&\mathrm{EU}_T^1(x, y \mid h_{T-1}) \\
&= \sum_{\lambda \in \Lambda} p(\lambda \mid h_{T-1}) \sum_{i,j \in A} x(i \mid h_{T-1}) y^\lambda(j \mid h_{T-1}) r^1(i,j) \\
&\mathrm{EU}_T^{2\lambda}(x, y \mid h_{T-1}) \\
&= \sum_{i,j \in A} x(i \mid h_{T-1}) y^\lambda(j \mid h_{T-1}) r^{2\lambda}(i,j), \qquad \forall \lambda \in \Lambda
\end{aligned}$$

Then, we consider time step $t+1 < T$, with history h_t.

$$\begin{aligned}
&\mathrm{EU}_{t+1}^1(x, y \mid h_t) = \sum_{\lambda \in \Lambda} \sum_{i,j \in A} p(\lambda \mid h_t) x(i \mid h_t) y^\lambda(j \mid h_t) r^1(i,j) \\
&+ \sum_{\lambda \in \Lambda} \sum_{i,j \in A} p(\lambda \mid h_t) x(i \mid h_t) y^\lambda(j \mid h_t) \, \mathrm{EU}_{t+2}^1(x, y \mid (h_t, i, j)) \\
&\mathrm{EU}_{t+1}^{2\lambda}(x, y \mid h_t) = \sum_{i,j \in A} x(i \mid h_t) y^\lambda(j \mid h_t) r^{2\lambda}(i,j) \\
&+ \sum_{i,j \in A} x(i \mid h_t) y^\lambda(j \mid h_t) \, \mathrm{EU}_{t+2}^{2\lambda}(x, y \mid (h_t, i, j)), \quad \forall \lambda \in \Lambda
\end{aligned}$$

We are now ready to introduce PBNE. Roughly speaking, PBNE requires both players to choose strategies that maximize their own expected utility starting from every possible history.

Definition 1 *A PBNE in a finitely repeated Bayesian game is a pair of strategies (x^*, y^*) such that for every history $h_t \in H \setminus H_T$,*

$$\begin{aligned}
\mathrm{EU}_t^1(x^*, y^* \mid h_t) &\geq \mathrm{EU}_t^1(x', y^* \mid h_t), \forall x' \\
\mathrm{EU}_t^{2\lambda}(x^*, y^* \mid h_t) &\geq \mathrm{EU}_t^{2\lambda}(x^*, y' \mid h_t), \forall y', \forall \lambda
\end{aligned}$$

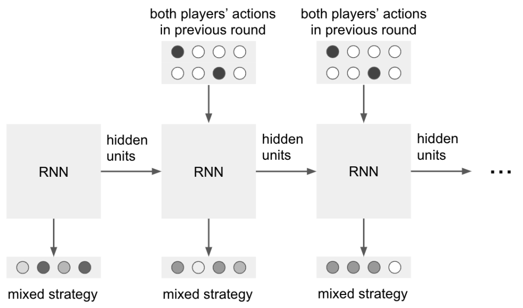


Figure 1: Network structure for RNN-based approach.

Nguyen et al. (2019) compute a PBNE is by using a mathematical program to encode all the utility constraints in the definition of PBNE. While this leads to, in theory, an exact solution, the algorithm can hardly scale, because the number of possible action histories grows exponentially with the size of the action space and time horizon. In the sequel, we present a learning-based approach to approximately compute a PBNE.

Two remarks are in order. First, although as a preliminary work we only consider one-sided incomplete information, the setup and our algorithmic results below can easily be extended to incomplete information on both sides. Similarly, extending the framework to stochastic games, where the stage game at each round could be different, also does not pose much technical difficulty.

4 Algorithm

Our learning-based approach is based on the the policy gradient (PG) algorithm in RL (Sutton et al. 2000). Let π be a policy parameterized by θ, and maps from a description of the current state and an action to a proability, i.e., $\pi_\theta(a|s)$ indicates the probability of taking action a given state (or state description) s following policy π_θ. PG has its foundation in the policy gradient theorem

$$\nabla_\theta \mathbb{E}_\pi[r] = \mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)]$$

where r is the reward for taking action a at state s where (s,a) is sampled from the distrbution induced by π, and Q^π is the state-value function of policy π.

The Recurrent Neural Network (RNN) (Werbos 1990) is a generalization of feedforward networks to sequences. It serves as a useful function approximator for domains with sequences and has been applied to neural language processing (Sutskever, Vinyals, and Le 2014) and RL (Wierstra et al. 2010), among others. RNN is a natural fit for this problem, as the action history takes form of a long sequence, which fits into the RNN structure perfectly. However, it fails to handle the exponentially increasing game tree and can have inconsistent subgame solution qualities, which contradicts our pursuit for PBNE. Therefore, we propose Approximated Temporal-Induced Neural Self-Play, which is capable of handling the exponential game tree with linear cost by using a belief-based representation and have consistent solution qualities in all subgames by using a backward induction styled learning scedule.

In the rest of the section, we first propose a simple implementation of the RNN-based approach and then present our proposed algorithm Approximated Temporal-Induced Neural Self-Play in detail, explaining its advantages over the RNN-based approach.

4.1 RNN-based approach

Player 1's policy network takes as input both players' action history and output the strategy in the next round (a probability distribution over the set of actions $\mathbf{A}$). We use the one-hot representation of both players actions (a^1_τ, a^2_τ) in the previous round and feed it to a RNN network as the τ-th input. Then we feed the hidden state in the last round $\mu_{\tau-1}$ into a feed-forward network and apply a softmax layer to μ_t to get the required distribution over actions for the $(t+1)$-th round. That is,

$$\mu_\tau = \text{RNN}(\mu_{\tau-1}, \text{one-hot}(a^1_\tau), \text{one-hot}(a^2_\tau)) \quad (1)$$

$$x(\cdot|h_t) = \text{Softmax}(\text{feed-forward}(\tau_t)) \quad (2)$$

The network structure is shown in Figure 1. Player 2's network has a very similar structure, except that there is an additional input λ indicating player 2's type.

We use *policy gradient self-play* to train both agents (Alg. 1). In each training iteration, we fix one of the agents and optimize another using policy gradient in turn. In the end, we use the time-averaged strategy as the final mixed strategy for each player. Instead of directly averaging over all the strategies in each round, We further approximate the time-averaged strategy by using the time-averaged network weights for the actor network. Empirically, such training approach yields strategies close to Nash equilibrium in our tested game setting.

Algorithm 1: Policy gradient self-play

```
Initialize $\theta^1_0, \theta^2_0$;
for $k = 1, 2, \ldots, maxIter$ do
    $\theta^1_k \leftarrow$ PolicyGradientForPlayer1$(\theta^1_{k-1}, \theta^2_{k-1})$;
    $\theta^2_k \leftarrow$ PolicyGradientForPlayer2$(\theta^2_{k-1}, \theta^1_k)$;
return $\frac{1}{maxIter}\sum_k \theta^1_k$, $\frac{1}{maxIter}\sum_k \theta^2_k$,
```

4.2 Belief-based approach

Although RNN-based approach seems natural for this game, it fails to address some major challenges of the game.

The structure of RNN is capable of dealing with long action history. However, it comes with the cost of increasing training resources. Generally, larger state space requires larger networks, larger training batch and more training episodes. The relationship between the action space and the training parameters to ensure the same level of solution quality is very hard to argue, let alone the potential need to change other scale-invariant hyperparameters such as learning rate.

On the other hand, we are trying to find the PBNE, which is a refinement of NE that requires optimal strategies in *all* subgames. In the RNN-based approach, the training variances in less visited subgames are higher, which can potentially lead to unbalanced and even unboundedly low solution quality in different subgames.

We can take another look at the game itself. To achieve PBNE, the agents need to make Nash equilibrium strategies for every subgame which is rooted at action history h_t. This is where the scalability issue comes up — there are exponentially many subgames to solve. To address this problem, rather than finding the optimal solution for every action history h_t, we can instead ask the agents to make decisions according to the belief of attacker's type distribution at each round of the game. Since we are dealing with finite attacker type set, the belief can be represented as a $|\mathbf{\Lambda}|$-dimensional vector in the probability simplex $\Delta^{|\mathbf{\Lambda}|}$. Now the agents need to find a mapping from $\Delta^{|\mathbf{\Lambda}|}$ to strategy space $\Delta^{|\mathbf{A}|}$. Despite that the state space grows from a finite set to a continuous space, we believe it is much easier for neural networks to generalize.

To get the belief at any round, we use Bayes' rule to update the belief step by step along the way.

$$b_{t+1}^{\lambda} = \frac{\Pr[a_t^{\lambda} = a_t^2] b_t^{\lambda}}{\sum_{\lambda'} \Pr[a_t^{\lambda'} = a_t^2] b_t^{\lambda'}}, \tag{3}$$

where a_t^{λ} is the random variable indicating the action that player 2 with type λ would take in round t, a_t^2 is the observed action of player 2, and b_t^{λ} is the probability that the player 2 is of type λ given its actions up to round $t-1$.

However, to do so, we require player 2 to reveal its probability of choosing a certain action, which is not a standard practice in RL. Normally, revealing one player's strategy may cause unfairness issue. Since we are using self-play to find the Equilibrium strategy, we do not need to take fairness into consideration in the training phase. During evaluation, we need to convert the belief-based strategy to action history based strategy. Suppose the result strategy is $\pi^1, \pi^{2\lambda} : \Delta^{|\mathbf{\Lambda}|} \mapsto \Delta^{|\mathbf{A}|}$, we can generate the action history based strategy $\hat{\pi}^1, \hat{\pi}^{2\lambda} : \mathcal{H} \mapsto \Delta^{|\mathbf{A}|}$ with the following procedure.

Algorithm 2: Conversion from belief-based strategy $\pi^1, \pi^{2\lambda}$ to action history based strategy $\hat{\pi}^1, \hat{\pi}^{2\lambda}$.

```
Function Convert(b, h)
    Data: $\mathbf{b} = \{b^{\lambda}\}$, current belief
    Data: $h = \{(a_1^1, a_1^2), \dots, (a_t^1, a_t^2)\}$, current
        action history
    $\hat{\pi}^1(h) \leftarrow \pi^1(\mathbf{b})$;
    $\hat{\pi}^{2\lambda}(h) \leftarrow \pi^{2\lambda}(\mathbf{b})$ for every $\lambda$;
    for $(a^1, a^2) \in \mathbf{A} \times \mathbf{A}$ do
        $\mathbf{b}' \leftarrow$ New belief assuming action $a^2$ is taken
          by player 2;
        Convert($\mathbf{b}'$, $h + (a^1, a^2)$);
Convert($\mathbf{p}$, $\{\}$);
```

4.3 Backward induction

Since our objective is to find an approximate PBNE in finitely repeated Bayesian games, instead of training the whole game in a top-down manner where we train each subgame only when we visited them, we introduce a *backward induction* training paradigm in which we train the problem in a bottom up manner.

In backward induction training, we start training in the bottom level subgame where there is only one round left. The goal here is to train a subgame policy that maps any belief to a strategy. Once the bottom level subgame policy is trained, we move up a level to train for the subgames that have two rounds left. However, when doing so, we only train for the strategy at the beginning of that subgame. When we collect experiences, for later games, we use previously trained subgame policies to continue the simulation, then sum up the all the rewards until the game ends and use that cumulative reward to optimize our training policy. We denote the subgames where there are k rounds left subgame-k. Suppose we are training subgame-k and all later subgame-k' where $k' < k$ has been trained, with the optimal policy denoted as $\pi_{k'}^*$. The policy that we are training is π_k. In the training of subgame-k, we are playing a surrogate game with utility function $\hat{u}$.

$$\hat{u}(a^1, a^{2\lambda}) = u(a^1, a^{2\lambda}) + \sum_{k'=1}^{k-1} u(a_{k'}^{1*}, a_{k'}^{2\lambda *}) \tag{4}$$

where $a_{k'}^{1*}$ and $a_{k'}^{2\lambda *}$ are sampled according to $\pi_{k'}^{1*}$ and $\pi_{k'}^{2\lambda *}$ and the updating belief along the way. We also use Policy Gradient self-play to solve the surrogate game.

4.4 Non-parametric policy approximation

In practice, we propose a *non-parametric approximation* method for training a policy that takes input any belief distribution. For each subgame-k, we first sample K belief vectors $\{\mathbf{b}_1, \dots, \mathbf{b}_K\}$ from $\Delta^{|\mathbf{\Lambda}|}$. For each $\mathbf{b}_K$, we use Policy Gradient self-play to find the optimal policy π_K with fixed type distribution $\mathbf{b}_K$. Then we add the belief strategy pair $(\mathbf{b}_K, \pi_K)$ to a strategy pool $\mathcal{D}_k$.

For a new belief $\mathbf{b}$ that is not sampled before, we use a distance based method to approximate the appropriate strategy for that belief, stated as follows:

$$\pi_k^*(\mathbf{b}, \cdot) \equiv \frac{\sum_{(\mathbf{b}', \pi) \in \mathcal{D}_k} w(\mathbf{b}, \mathbf{b}') \pi(\cdot)}{\sum_{(\mathbf{b}', \pi) \in \mathcal{D}_k} w(\mathbf{b}, \mathbf{b}')} \tag{5}$$

where

$$w(\mathbf{b}, \mathbf{b}') \equiv \frac{1}{\max(\varepsilon, \|\mathbf{b} - \mathbf{b}'\|^2)} \tag{6}$$

5 Experiments

5.1 Security Game

We test our algorithm in finitely repeated *simultaneous-move* security game as discussed in (Nguyen et al. 2019). In this game, the behavior strategy profile for the defender is a mapping from past action history h_t to a probability distribution over different resource distributions $\mathbf{s}, \mathbf{x} = \{x(\mathbf{s}|h_t) : \sum_{\mathbf{s}} x(\mathbf{s}|h_t) = 1, x(\mathbf{s}|h_t) \in [0,1]\}$. The strategy profile for the attacker is a mapping from past action history h_t to a probability distribution over different targets for each type $\lambda, \mathbf{y}^\lambda = \{y^\lambda(i|h_t) : \sum_i y^\lambda(i|h_t) = 1, y^\lambda(i|h_t) \in [0,1]\}$.

Evaluation Since this game is general-sum and can have multiple PBNEs, we evaluate our solution quality by calculating the difference between the expected utility when playing against each other and the expected utility of the best response strategy to each player. Formally, for defender strategy $\mathbf{x}$ and attacker strategy $\mathbf{y} = \{\mathbf{y}^\lambda\}$, the solution quality ε is defined as

$$\varepsilon = \max\big\{ \mathrm{EU}(\mathrm{BR}(\mathbf{y}), \mathbf{y}) - \mathrm{EU}(\mathbf{x}, \mathbf{y}), \\ \max\{\mathrm{EU}^\lambda(\mathbf{x}, \mathrm{BR}^\lambda(\mathbf{x})) - \mathrm{EU}^\lambda(\mathbf{x}, \mathbf{y}^\lambda)\}\big\}$$

We also check the PBNE quality by taking the maximum difference in all subgames.

Game Setting We limit defender resources $K = 1$ and test our algorithm on various number of targets $|\mathbf{N}|$, number of types $|\mathbf{\Lambda}|$ and number of time steps $|\mathbf{T}|$.

Throughout our experiments, we set defender resources to $K = 1$ and number of attacker types to $|\mathbf{\Lambda}| = 2$. Attacker and defender's rewards and penalties are generated uniformly randomly from $[1, 10]$ and $[-10, -1]$ respectively.

5.2 Comparison with mathematical programming

We first compare our algorithm directly with the mathematical programming solution in (Nguyen et al. 2019). We show that our algorithm can deal with the increase of numbers of targets $|\mathbf{N}|$ and number of time steps $|\mathbf{T}|$ comfortably with linear training time and space usage and linear solution quality loss while the mathematical programming method suffers exponential cost increase both in time and space.

To showcase the scalability, we tested our algorithm in $|\mathbf{N}| = 2$ targets game with time step $|\mathbf{T}| \in \{1, 2, \ldots, 10\}$ and in $|\mathbf{N}| = 5$ targets game with time step $|\mathbf{T}| \in \{1, 2, \ldots, 5\}$. For comparison, the mathematical programming solution runs out of memory in 2 targets game when $|\mathbf{T}| \geq 7$ and in 5 targets game when $|\mathbf{T}| \geq 4$.

Each entry in the table for our method is the averaged difference over 11 different priors, $(0.0, 1.0), (0.1, 0.9), \ldots, (1.0, 0.0)$ in 4 different game instances. The result is in Table 1.

5.3 Result analysis

Here we provide several tests to demonstrate the soundness of our result.

Variance test We test our method on 4 different game instances. We provide the solution quality (measured by utility difference to best response)-$|\mathbf{T}|$ curve to show that our result has a good consistency among different game seeds. The result is in Figure 2.

$\|\mathbf{N}\|$	$\|\mathbf{T}\|$	Baseline	Ours
2	1	$< 10^{-8}$	0.112
2	2	$\approx 10^{-8}$	0.224
2	3	$\approx 10^{-7}$	0.354
2	4	$\approx 10^{-6}$	0.470
2	5	$\approx 10^{-5}$	0.570
2	6	$\approx 10^{-5}$	0.681
2	7	N/A	0.769
2	8	N/A	0.909
2	9	N/A	0.994
2	10	N/A	1.130
5	1	$\approx 10^{-6}$	0.254
5	2	$\approx 10^{-6}$	0.531
5	3	$\approx 10^{-3}$	0.839
5	4	N/A	1.259
5	5	N/A	1.804

Table 1: Solution quality comparison between baseline method and ours.

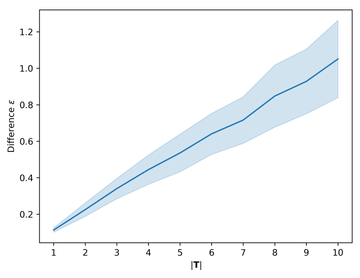
PBNE test To demonstrate that our algorithm is capable of finding a *Perfect Bayesian Nash Equilibrium*, we record our solution quality in every subgame rooted on action history $\mathbf{h}_t$. We then take the maximum difference in all $\sum_{t=0}^{|\mathbf{T}|} |\mathbf{N}|^t$ subgames and divide it by the overall difference. Notice that the overall difference is also a subgame (with action history $\{\}$) difference, which guarantees the ratio to be no less than 1. The result is shown in Figure 3. We observe that with 95% confidence the solution quality in every subgame is no worse than 10% of the overall solution quality.

Convergence test We run the Policy Gradient self-play algorithm on two sample settings and record the solution quality curve against training iterations. We observe that in both cases the difference in expected utilities are gradually reducing. However, in the second case, the lower bound of the difference is bounded by the results in previously trained policies.

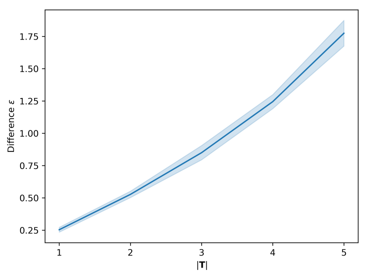
5.4 Attacker deception analysis

We also analyze the attacker's deception behaviour in our result. We quantify attacker's incentive to deceive by calculating the KL divergence between a PBNE attacker's strategy and a myopic attacker's strategy. The result is shown in Figure 5. From the figure we can see that more rounds and less prior probability can both boost the attacker's incentive to deceive. We also show the averaged KL divergence for different $|\mathbf{T}|$ to demonstrate the increasing incentive to deceive with more time steps in Figure 5c. We can also observe that the growth rate of KL divergence gradually slows down. Which gives us a hint that for even longer games, the strategy for each round may converge.

Furthermore, we introduce another parameter - β, to manipulate the attacker's incentive to deceive. In the i-th turn of the game, the reward (utility) for both players in that turn will be multiplied by β^{i-1}. ($i = 1, 2, \ldots, |\mathbf{T}|$) Our normal setting corresponds to $\beta = 1$ and myopic playing (stage game equilibrium) corresponds to $\beta = 0$. Larger β will de-

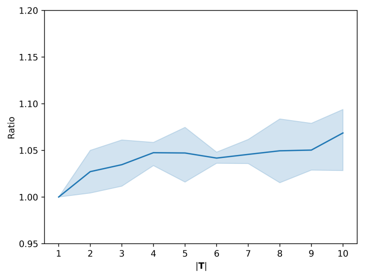


(a) 2 Targets

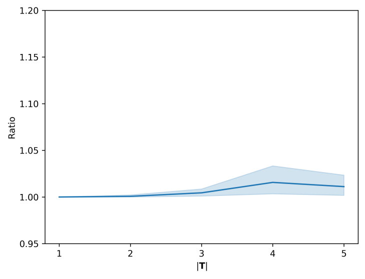


(b) 5 Targets

Figure 2: Solution quality across 4 game instances. Y-axis is the solution quality measured by difference in expected utility to best response and X-axis is number of time steps $|\mathbf{T}|$. The shadow represents the standard deviation over different game instances to show the variance of our method.



(a) 2 Targets.



(b) 5 Targets.

Figure 3: Subgame difference / overall difference ratio. We observe that in both 2-target games and 5-target games our method suffers less than 10% quality loss with 95% confidence.

crease the significance of early rounds in the long run and therefore increase the attacker's incentive to deceive.

We tested our algorithm on $\beta = 1, 1.5, 2$ and compare our result with the myopic strategy ($\beta = 0$). The result is shown in Figure 6. We can clearly see from the figure that larger β leads to more deviation from the myopic strategy, which matches our assertion.

5.5 Training details

We use PPO (Schulman et al. 2017) for the Policy Gradient part. we use Adam optimizer with learning rate 10^{-3}, $\beta_1 = 0$, $\beta_2 = 0.99$. The choice of β_1 is to accommodate the non-stationary environment (opponent's strategy). The batch size of 100. For each subgame, we train the network for 1000 iterations. All the experiments are run on a personal laptop with Intel® Core$^{\text{TM}}$ i7-9750H CPU (2.60GHz, 12 cores) and a single GeForce GTX 1660 Ti graphics card.

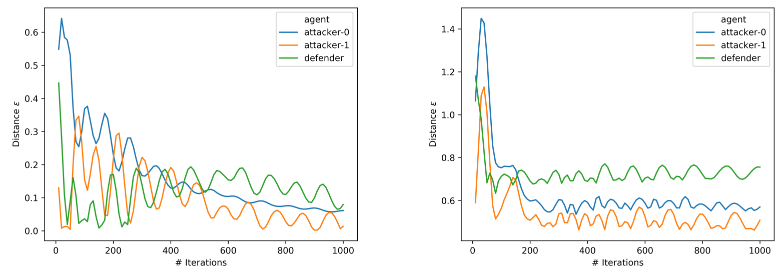
6 Conclusion

In this paper, we propose a belief-based backward induction fashioned reinforcement learning algorithm to solve the *two-player general-sum repeated Bayesian games*. Compared to mathematical programming, our approach resolve the scalability problem at a tolerable cost of solution quality while preserving the ability to find PBNE, a refinement of NE, which requires optimal solution in every subgames.

This paper presents a preliminary result of our work. We seek to apply this approach in more complicated games which can fit into the two-player finitely repeated Bayesian game framework. For example, the deception game in (Lowe et al. 2017). We hope to train intelligent agents that have the mindset of orchestrating its behaviour with regard to the information difference in a complicated environment in the future.

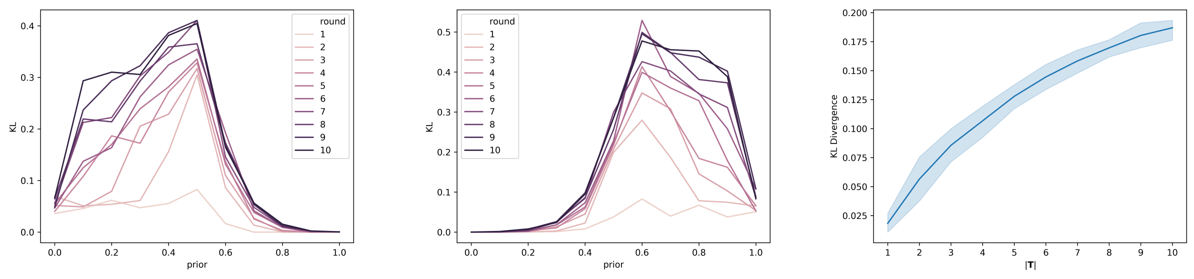
7 Acknowledgment

Co-author Fang is supported in part by NSF grant IIS-1850477, a research grant from Lockheed Martin. Co-authors Fang and Shi are supported in part by the U.S. Army Combat Capabilities Development Command Army Research Laboratory Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the funding agencies.



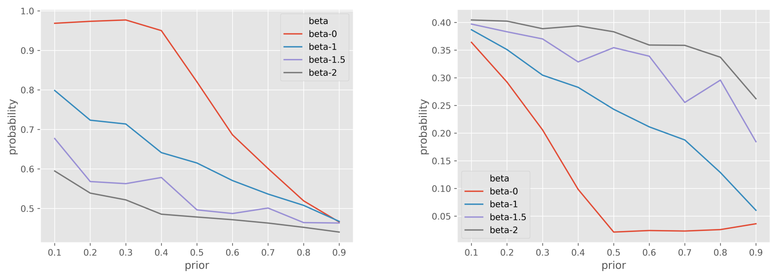
(a) $|\mathbf{N}| = 2$, $|\mathbf{T}| = 1$, $\mathbf{b} = (0.3, 0.7)$ (b) $|\mathbf{N}| = 2$, $|\mathbf{T}| = 10$, $\mathbf{b} = (0.3, 0.7)$

Figure 4: Convergence curve of Policy Gradient self-play. The three curves in each figure represents the solution quality of the time averaged policy of type-1 attacker, type-2 attacker and defender. (a): Result on subgame-1 in 2-target game with belief $(0.3, 0.7)$. (b): Result on subgame-10 in 2-target game with belief $(0.3, 0.7)$. Notice that in (b) the solution quality is affected by the cumulative error in previous trained policies which makes it impossible to approach zero.



(a) Attacker type 1 (b) Attacker type 2 (c) Average

Figure 5: Attacker's strategies' KL divergence with myopic strategy when $|\mathbf{N}| = 2$. (a), (b): KL divergence curves for each Attacker types. Each curve represents a $|\mathbf{T}|$. The X-axis is the prior probability of type-1. (c): Averaged KL divergence over all attacker types and prior distributions.



(a) Attacker type 1 (b) Attacker type 2

Figure 6: Attacker's strategies with different βs. All tests are done with time step $|\mathbf{T}| = 5$ and targets $|\mathbf{N}| = 2$. Both plots show the probability of the attacker choosing target 1, against different prior distributions. (the X-axis corresponds to the prior probability of type-1)

References

Albrecht, S. V., and Ramamoorthy, S. 2013. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. In *AAMAS*, 1155–1156. AAMAS.

An, B.; Tambe, M.; Ordonez, F.; Shieh, E.; and Kiekintveld, C. 2011. Refinement of strong stackelberg equilibria in security games. In *AAAI*.

Bielefeld, R. S. 1988. Reexamination of the perfectness concept for equilibrium points in extensive games. In *Models of Strategic Rationality*. Springer. 1–31.

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

Chandrasekaran, M.; Chen, Y.; and Doshi, P. 2017. On markov games played by bayesian and boundedly-rational players. In *AAAI*.

Cho, I.-K., and Kreps, D. M. 1987. Signaling games and stable equilibria. *The Quarterly Journal of Economics* 102(2):179–221.

Etessami, K.; Hansen, K. A.; Miltersen, P. B.; and Sørensen, T. B. 2014. The complexity of approximating a trembling hand perfect equilibrium of a multi-player game in strategic form. In *International Symposium on Algorithmic Game Theory*, 231–243. Springer.

Farina, G., and Gatti, N. 2017. Extensive-form perfect equilibrium computation in two-player games. In *AAAI*.

Farina, G.; Kroer, C.; and Sandholm, T. 2017. Regret minimization in behaviorally-constrained zero-sum games. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 1107–1116. JMLR. org.

Forges, F. 1992. Repeated games of incomplete information: non-zero-sum. *Handbook of game theory with economic applications* 1:155–177.

Hansen, K. A., and Lund, T. B. 2018. Computational complexity of proper equilibrium. In *EC*, 113–130. ACM.

Harsanyi, J. C. 1968. Games with incomplete information played by "bayesian" players part ii. bayesian equilibrium points. *Management Science* 14(5):320–334.

Heinrich, J.; Lanctot, M.; and Silver, D. 2015. Fictitious self-play in extensive-form games. In *ICML*, 805–813.

Hu, J.; Wellman, M. P.; et al. 1998. Multiagent reinforcement learning: theoretical framework and an algorithm. In *ICML*, volume 98, 242–250. Citeseer.

Kamra, N.; Gupta, U.; Wang, K.; Fang, F.; Liu, Y.; and Tambe, M. 2019. Deepfp for finding nash equilibrium in continuous action spaces. In Alpcan, T.; Vorobeychik, Y.; Baras, J. S.; and Dán, G., eds., *Decision and Game Theory for Security*, 238–258. Cham: Springer International Publishing.

Kreps, D. M., and Wilson, R. 1982. Sequential equilibria. *Econometrica: Journal of the Econometric Society* 863–894.

Kroer, C.; Farina, G.; and Sandholm, T. 2017. Smoothing method for approximate extensive-form perfect equilibrium. *arXiv preprint arXiv:1705.09326*.

Littman, M. L. 1994. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*. Elsevier. 157–163.

Lowe, R.; Wu, Y.; Tamar, A.; Harb, J.; Abbeel, P.; and Mordatch, I. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. *NIPS*.

Miltersen, P. B., and Sørensen, T. B. 2010. Computing a quasi-perfect equilibrium of a two-player game. *Economic Theory* 42(1):175–192.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature*.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 1928–1937.

Nesterov, Y. 2005. Excessive gap technique in nonsmooth convex minimization. *SIAM Journal on Optimization* 16(1):235–249.

Nguyen, T. H.; Wang, Y.; Sinha, A.; and Wellman, M. P. 2019. Deception in finitely repeated security games. *AAAI* 33:2133–2140.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *arXiv e-prints* arXiv:1707.06347.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science* 362(6419):1140–1144.

Sorin, S. 2003. Stochastic games with incomplete information. In *Stochastic Games and applications*. Springer. 375–395.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to Sequence Learning with Neural Networks. *NeurIPS* 3104–3112.

Sutton, R. S.; McAllester, D. A.; Singh, S. P.; and Mansour, Y. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, 1057–1063.

Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature* 1–5.

Werbos, P. 1990. Backpropagation through time: what it does and how to do it. *Proceedings of IEEE*.

Wierstra, D.; Förster, A.; Peters, J.; and Schmidhuber, J. 2010. Recurrent policy gradients. *Logic Journal of the IGPL* 18:620–634.

Zinkevich, M.; Johanson, M.; Bowling, M.; and Piccione, C. 2008. Regret minimization in games with incomplete information. In *NeurIPS*, 1729–1736.