Randomized Solutions to Partial Information Dynamic Zero-Sum Games

Shaunak D. Bopardikar  João P. Hespanha

Abstract—This paper presents randomized methods to solve partial information dynamic zero-sum games. We extend the recently introduced sampled saddle-point (SSP) algorithm, which provided probabilistic security guarantees in static zero-sum matrix games. A straightforward extension to partial information dynamic games is to apply the SSP algorithm to the matrix obtained by recording the outcomes of playing every policy of one player against every policy of the other player. However, the matrix so obtained has typically a very large size. This paper formalizes a novel extension of the SSP algorithm to partial information dynamic games, which does not require generating the entire matrix. We show that the bounds derived for the SSP algorithm in the static case, provide the same level of probabilistic security for a partial information dynamic game. The effectiveness of the procedure is demonstrated by solving a prototypical example of a board game with partial information, for which no deterministic security levels have been published.

Index Terms—Dynamic Games, Randomized Algorithms, Partial Information

I. INTRODUCTION

A game is termed to be dynamic if it has a state that evolves as a function of the actions of the players. Dynamic games provide an effective framework to model problems in diverse areas such as surveillance, network security, operations research and economics. For full information games, such as Chess, in which both players have access to the entire state, dynamic programming (cf. [1]) is an efficient method to compute optimal strategies for the players. However, for partial information games, such as Poker, wherein each player must hypothesize on the past moves of the opponent, the method often leads to solving very large matrix games making the approach computationally intractable [2].

Recently, randomized methods were proposed in [3] to solve zero-sum matrix games for large-sized matrices. Each player reduces her search space by taking a random sub-matrix to construct a much smaller version of the original game. Players then solve these smaller sub-matrix games and utilize the saddle-point policies so obtained against each other. We showed that when the size of the sub-matrix is sufficiently large, there is only a small probability that the outcome of the game is significantly worse than the outcome anticipated by a player. This approach leads to probabilistic notions of security values and security policies for the game. While these techniques and results were focussed on single stage matrix games, this paper formalizes the approach from [3] to partial information dynamic games.

A classic approach to evaluate dynamic games is to write the game in an extensive “tree” form, and then evaluate the tree recursively to obtain the optimal strategies and the value of the game [1]. However, this approach turns computationally unattractive when the number of stages or the action space is large. Imposing a partial information structure further adds significant complexity to the problem.

Randomized methods have been successful in providing efficient solutions to complex control design problems with probabilistic guarantees. [4] adopts a probabilistic approach to show the existence of randomized algorithms with polynomial complexity to solve complex robust stability analysis problems. [5] proposes a randomized method to determine the minimum number of samples that provide a probabilistic guarantee of the level of worst-case controller performance. In [6], [7], the authors demonstrate the use of randomized algorithms in statistical learning theory to solve control design problems and a number of well known complex problems in matrix theory. In [8], [9], the authors introduce the scenario approach to solve convex optimization problems with an infinite number of constraints, and discuss possible applications of the approach to systems and control. In [10], the authors provide an improvement in sample-size bounds and also over the bounds on the scenario approach for convex optimization. In [3], we used these results to provide bounds on the size of the subgames for probabilistic security.

There have been recent efforts to provide efficient solutions to partial information games. [11] presents a novel representation of the game tree which reduces the complexity of finding optimal mixed strategies to linear in the size of the game tree. [12] characterizes initial information state of a stochastic imperfect information game, for which the information state propagation is tractable. [13] presents idempotent methods for dynamic games.

This paper extends the recently introduced sampled saddle-point (SSP) algorithm [3], which provides probabilistic security guarantees in static zero-sum matrix games, to the class of partial information dynamic games. A straightforward extension to partial information dynamic games is to apply the SSP algorithm on a matrix obtained by recording the outcomes of playing every policy of one player against every policy of the other player, for every compatible information structure. In [2], this procedure was used to show existence of Nash equilibrium among stochastic behavioral policies in Markov games, but not as a computational tool. However, the matrix so obtained has typically a very large size. The novelty of our extension of the SSP algorithm to partial information dynamic games is that the procedure proposed here does not require generating the entire matrix.
We show that the bounds derived for the SSP algorithm in the static case, provide the same level of probabilistic security for a partial information dynamic game.

To demonstrate the effectiveness of this general algorithm, we consider a prototypical example of a Tic-Tac-Toe game with partial information, i.e., neither player can view the entries in certain squares. The complexity of such a version grows rapidly with the size of the game, and the number of “hidden” squares considered. However, to the best of our knowledge, even for a simple three-by-three version with three hidden squares, deterministic security policies and levels have not been published. We apply our general algorithm to compute probabilistic levels of security for this game. Our approach is independent of the size of the game and depends only on the size of the subsets that each player looks at, in order to arrive at her strategy. This is possible because each player concentrates on a subset of her policy set, and probabilistic guarantees rather than deterministic guarantees on the quality of the solution with respect to the actual game outcome are considered.

This paper is organized as follows. The SSP algorithm and the related main results are reviewed in Section II. The version of the SSP algorithm for dynamic games and the related results are presented in Section III. An extension to policy domination in dynamic games is considered in Section IV. Finally, we demonstrate the procedure applied to a partial information Tic-Tac-Toe game in Section V.

II. REVIEW: SAMPLED SADDLE-POINT ALGORITHM

In the Sampled Saddle-Point (SSP) algorithm [3], each player samples a sub-matrix from the original matrix, solves a smaller game and utilizes the saddle-point policy so obtained against the other. Specifically, if $B^{k \times l}$ denotes the set of $k \times l$ left-stochastic $(0,1)$-matrices (i.e., matrices whose entries belong to the set $\{0,1\}$ and whose columns add up to one), then the SSP algorithm can be summarized as follows.

**Algorithm 1: Sampled Saddle-Point Algorithm**

1. For $P_1$: Select random matrices $\Gamma_1 \in B^{M \times m_1}$, $\Pi_1 \in B^{N \times n_1}$
2. Compute sub-matrix: $A_1 = \Gamma_1 \Pi_1$
3. Security policy: $y_1^* \in \text{argmin}_{y_1 \in S_{m_1}} \max_{z \in S_{n_1}} y_1^T A_1 z$
4. Security value: $V(A_1) = \max_{z \in S_{n_1}} y_1^T A_1 z$
5. For $P_2$: Select random matrices $\Gamma_2 \in B^{M \times m_2}$ and $\Pi_2 \in B^{N \times n_2}$
6. Compute sub-matrix: $A_2 = \Gamma_2 \Pi_2$
7. Security policy: $z_2^* \in \text{argmax}_{z_2 \in S_{n_2}} \min_{y_2 \in S_{m_2}} y_2^T A_2 z_2$
8. Security value: $V(A_2) = \min_{y_2 \in S_{m_2}} y_2^T A_2 z_2^*$

**Output:** $y_1^T A_1 z_2^* := y_1^* \Gamma_1 \Pi_2 z_2^*$

The SSP algorithm is $\epsilon$-secure for player $P_1$ with confidence $1 - \delta$ if

$$P_{\Gamma_1, \Pi_1, \Gamma_2, \Pi_2} \left( y_1^* A_1 z_2^* \leq \tilde{V}(A_1) + \epsilon \right) \geq 1 - \delta.$$ 

The subscript in the probability measure $P$ emphasizes which random variables define the event that is being measured. To provide guarantees for specific policies/values, the following notions of security that refer to specific policies/values are introduced. The policy $y_1^*$ with value $\tilde{V}(A_1)$ is $\epsilon$-secure for player $P_1$ with confidence $1 - \delta$ if

$$P_{\Gamma_1, n_1} \left( y_1^* A_1 z_2^* \leq \tilde{V}(A_1) + \epsilon, \tilde{V}(A_1) \right) \geq 1 - \delta.$$ 

The first result provides a bound on the size of the sub-matrices for the players that guarantees $\epsilon$-security with $\epsilon = 0$.

**Theorem II.1 (A-priori bounds, [3], [14])** Suppose that the matrices $\Gamma_1, \Pi_1, \Gamma_2, \Pi_2$ are statistically independent. If $\Pi_1$ and $\Pi_2$ have identically distributed columns and

$$n_1 = \left[ \frac{m_1 + 1}{\delta} - 1 \right] n_2, \tag{1}$$

for some $n_2 \geq n_2$, then the SSP algorithm is $\epsilon = 0$-secure for $P_1$ with confidence $1 - \delta$. Further, if

$$n_1 = \left[ \frac{1}{\delta} \left( \ln \frac{1}{\beta} + m_1 + \sqrt{2m_1 \ln \frac{1}{\beta}} \right) \right] n_2, \tag{2}$$

then, with probability $\epsilon$ higher than $1 - \beta$, the policy $y_1^*$ with value $\tilde{V}(A_1)$ is $\epsilon = 0$-secure for $P_1$ with confidence $1 - \delta$.

Suppose that, due to computational limitations, player $P_1$ cannot satisfy the above bounds. Then, one option to maintain the same high level of confidence $1 - \delta$ would be to accept a larger value for $\epsilon$. Algorithm 2 summarizes an a-posteriori procedure for $P_1$. Let $e_j(k_1)$ denote the $j$th element of the canonical basis of $\mathbb{R}^{k_1}$.

**Algorithm 2: A-posteriori procedure**

1. Pick some values for $m_1, n_1$
2. Determine $y^*$ and $\tilde{V}(A_1)$ using SSP algorithm
3. Randomly select $\Pi_1 \in B^{N \times n_1}$ using distribution of $\Pi_1$

**Output:** $\bar{v} := \max_{j \in \{1, \ldots, k_1\}} y_1^T \Pi_1 e_j(k_1)$

The following a-posteriori guarantees can be obtained.

**Theorem II.2 (A-posteriori bounds, [3], [14])** Suppose that the matrices $\Gamma_1, \Pi_1, \Gamma_2, \Pi_2$ are statistically independent. If $\Pi_1$ and $\Pi_2$ have identically distributed columns and

$$k_1 = \left[ \frac{1}{\delta} - 1 \right] n_2, \tag{3}$$

for some $n_2 \geq n_2$, then the SSP algorithm is $\epsilon$-secure for $P_1$ with confidence $1 - \delta$ for any

$$\epsilon \geq \bar{v} - \tilde{V}(A_1). \tag{4}$$

If one further increases $k_1$ to satisfy

$$k_1 = \left[ \frac{1}{\delta} \ln \frac{1}{\beta} \right] n_2, \tag{5}$$

then, with probability higher than $1 - \beta$, the policy $y_1^*$ with value $\tilde{V}(A_1)$ is $\epsilon$-secure for $P_1$ with confidence $1 - \delta$.

1. The confidence level $1 - \beta$ for $P_1$ refers solely to the extraction of the matrix $\Pi_1$ and holds for any given matrix $\Gamma_1$. 

4040
III. SSP Algorithm for Dynamic Games

In this section, we describe a version of the SSP algorithm, and extend the previous bounds for the class of dynamic games. We begin by formalizing the notion of policies in dynamic games. Let $\mathcal{I} \cup \emptyset$ denote the set of all information states, where $\emptyset$ denotes the empty set. Let $\mathcal{A}$ denote the set of all allowed actions. In a dynamic game, a \textit{completely defined policy} is a map $\gamma : \mathcal{I} \cup \emptyset \rightarrow \mathcal{A}$ that assigns an allowed action for every information state.

A naive version of the SSP algorithm is the following: Consider all possible completely defined policies $\{\gamma_1, \ldots, \gamma_M\}$ of $P_1$ and $\{\sigma_1, \ldots, \sigma_N\}$ of $P_2$. For every $(i, j) \in \{1, \ldots, M\} \times \{1, \ldots, N\}$, the entry $a_{ij}$ of matrix $A$ is the outcome of playing policy $\gamma_i$ of $P_1$ against $\sigma_j$ of $P_2$. The naive version of the SSP algorithm is now exactly identical to that in Section II, with a difference that the rows and the columns of the matrix $A$ now correspond to policies instead of actions, as in the case of static matrix games. Thus, Theorem II.1 holds for this naive version of the SSP.

However, the size of the matrix $A$ is potentially very large for dynamic games, especially when the entire information state is not accessible to either player. So, instead of generating the entire matrix and then sampling from the matrix, we will now present an incremental procedure wherein each player independently constructs their sub-matrix using the same set of rules, and then both players play their resulting sampled security policies against each other.

A. SSP algorithm with Incremental Matrix Construction

The main idea behind our approach for constructing the sampled sub-matrix is that each player plays some of her policies one-by-one against some policies which the opponent is anticipated to play, with each of the policies being undefined to begin with. Whenever a player encounters a new (hitherto unseen) information state, she picks an action using a stochastic heuristic. In this process, the policies of the player, as well as the anticipated policies of the opponent are defined partially, and the outcome of the game is recorded as an entry of the sampled sub-matrix. By a partially defined policy $\gamma$, we mean that for some information state $I \in \mathcal{I} \cup \emptyset$, $\gamma(I) = \emptyset$. In the end, a player obtains a partially defined set of policies and her sampled sub-matrix.

The SSP algorithm for dynamic games is described as follows, with a summary provided in Algorithms 3 and 4.

1) Player $P_1$ first initializes her policies $\{\gamma_1, \ldots, \gamma_{m_1}\}$ and the policies $\{\bar{\sigma}_1, \ldots, \bar{\sigma}_{n_1}\}$, which refer to $P_1$'s guess of the policies which $P_2$ will employ to play the game, to the empty set (Step 1 of Algorithm 3).

2) Player $P_1$ then generates her sub-matrix $A_1$ using Algorithm 4. This algorithm takes as inputs the number of policies $m_1$ and $n_1$, and the policies $\gamma$ and $\bar{\sigma}$ which have been initialized to the empty set.

3) Algorithm 4 begins by playing policy $\gamma_1$ against $\bar{\sigma}_1$. The information state $I$ is initialized to the empty set (Step 3). Whenever a new information state is encountered (Steps 5 and 8), $P_1$ plays a random action chosen using a stochastic heuristic $\text{Heur}_1$, and expects that $P_2$ will also play a random action chosen using a stochastic heuristic $\text{Heur}_2$, where given a probability space $(\Omega, \mathcal{F}, P)$, the heuristics are stochastic maps $\text{Heur}_1, \text{Heur}_2 : (\mathcal{I} \cup \emptyset) \times \Omega \rightarrow \mathcal{A}$. After a player plays her action, the information state $I$ is updated (Steps 7 and 10) using update functions $\text{Update}_1, \text{Update}_2 : (\mathcal{I} \cup \emptyset) \times \mathcal{A} \rightarrow \mathcal{I}$. When the game terminates, the outcome is set equal to the entry $a_{11}$ of matrix $A_1$.

4) Player $P_1$ then repeats this procedure (inner for loop) and plays the policy $\gamma_1$ against other policies $\{\bar{\sigma}_2, \ldots, \bar{\sigma}_{n_1}\}$, and the outcomes represent the first row of the sampled matrix $A_1$. The resulting policy $\gamma_1$ may not be completely defined.

5) Player $P_1$ repeats the procedure (of items 3 and 4 above) that define policies $\{\gamma_1, \ldots, \gamma_{m_1}\}$ at least partially (outer for loop of Algorithm 4). This procedure defines the sampled sub-matrix $A_1$.

6) Player $P_1$ computes the mixed security policy and the security value for her sub-matrix $A_1$ (Steps 3 and 4 of Algorithm 3) using

$$\bar{V}(A_1) = \max_{z \in \mathcal{S}_{m_1}} y^* A_1 z = \min_{y \in \mathcal{S}_{n_1}} \max_{z \in \mathcal{S}_{m_1}} y^* A_1 z,$$

where $\mathcal{S}_{m_1}$ and $\mathcal{S}_{n_1}$ denote the probability simplexes of appropriate dimensions. We call $y^*_1$ as the \textit{sampled mixed security policy} and $\bar{V}(A_1)$ as the sampled security value of the game for player $P_1$.

7) Player $P_2$ repeats the steps from items 1 – 6 to obtain her sampled policies $\{\sigma_1, \ldots, \sigma_{n_1}\}$, her sampled mixed security policy $z^*_2$ and her sampled security value $\bar{V}(A_2)$ (Steps 5 – 8 of Algorithm 3).

8) Player $P_1$ selects a policy out of $\{\gamma_1, \ldots, \gamma_{m_1}\}$ using the distribution $y^*_1$ and plays it against $P_2$’s policy chosen out of $\{\sigma_1, \ldots, \sigma_{n_1}\}$. Thus, the expected outcome of this procedure is equivalent to the following: the players generate the sub-matrix $A_{12}$ (Step 9 of Algorithm 3), i.e., play the policies $\gamma$ and $\sigma$ against each other using Algorithm 4. Then, the expected outcome is given by $y^*_1 A_{12} z^*_2$.

Consistent with Section II, we say that the above SSP algorithm is $\epsilon$-secure for player $P_1$ with confidence $1 - \delta$ if

$$P_{\text{Heur}_1, \text{Heur}_2}(y^*_1 A_{12} z^*_2 \leq \bar{V}(A_1) + \epsilon) \geq 1 - \delta.$$

We say that the \textit{mixed security policy} $y^*$ with value $\bar{V}(A_1)$ is $\epsilon$-secure for player $P_1$ with confidence $1 - \delta$ if

$$P_{\text{Heur}_1, \text{Heur}_2}(y^* A_{12} z^* \leq \bar{V}(A_1) + \epsilon | y^*, \bar{V}(A_1)) \geq 1 - \delta.$$

The random variables in these probability definitions are the heuristics $\text{Heur}_1$ and $\text{Heur}_2$.

Remark III.1 (Partial Information games) To generate sampled matrices in Algorithm 4, the players need not have access to the entire information state $I$. Therefore, as we are concerned only with the probability of unpleasant surprises, the above procedure extends also to the class of partial information games.
Algorithm 3: SSP for Dynamic Games

1. **P1 Initialization**: \( \gamma := [\gamma_1, \ldots, \gamma_{m_1}] = [\emptyset, \ldots, \emptyset] \), \( \bar{\sigma} := [\bar{\sigma}_1, \ldots, \bar{\sigma}_{n_1}] = [\emptyset, \ldots, \emptyset] \).

2. Generate sub-matrix: 
   \[ A_1, \gamma, \bar{\sigma} = \text{GenMatrix}(m_1, n_1, \gamma, \bar{\sigma}) \]

3. Security policy: \( y_1^* \in \arg\min_{y_1 \in S_{n_1}} \max_{z \in S_{m_1}} y_1' A_1 z \)

4. Security value: \( V(A_1) = \max_{y_1 \in S_{n_1}} y_1' A_1 z \)

5. **P2 Initialization**: \( \bar{\gamma} := [\bar{\gamma}_1, \ldots, \bar{\gamma}_{m_2}] = [\emptyset, \ldots, \emptyset] \), \( \sigma := [\sigma_1, \ldots, \sigma_{n_2}] := [\emptyset, \ldots, \emptyset] \).

6. Generate sub-matrix: 
   \[ A_2, \bar{\gamma}, \sigma = \text{GenMatrix}(m_2, n_2, \bar{\gamma}, \sigma) \]

7. Security policy: \( z_2^* \in \arg\max_{z_2 \in S_{n_2}} \min_{y \in S_{m_2}} y' A_2 z_2 \)

8. Security value: \( V(A_2) = \min_{y \in S_{m_2}} y' A_2 z_2^* \)

9. **Play sampled policies**: Generate sub-matrix 
   \[ A_{12}, \gamma, \bar{\sigma} = \text{GenMatrix}(m_1, n_2, \gamma, \bar{\sigma}) \]

Output: \( y_1^* A_{12} z_2^* \)

Algorithm 4: GenMatrix

Input: \( m, n, [\gamma_1, \ldots, \gamma_m], [\sigma_1, \ldots, \sigma_n] \)

1. foreach \( i = 1, \ldots, m \) do
   2. foreach \( j = 1, \ldots, n \) do
      3. \( I = \emptyset \)
      4. while \((i, j)-\text{th game not terminated}\) do
         5. if \( \gamma_i(I) = \emptyset \) then
            6. \( \gamma_i(I) = \text{Heur}_1(I, \omega_1) \)
            7. \( I = \text{Update}_1(I, \gamma_i(I)) \)
         8. if \( \sigma_j(I) = \emptyset \) then
            9. \( \sigma_j(I) = \text{Heur}_2(I, \omega_2) \)
            10. \( I = \text{Update}_2(I, \sigma_j(I)) \)

Output: \( A, [\gamma_1, \ldots, \gamma_m], [\sigma_1, \ldots, \sigma_n] \)

B. Probabilistic guarantees

Since both players use the same procedure to compute their respective sampled security policies, Algorithm 3 is equivalent to both players sampling rows and columns independently from identical distributions from the matrix \( A \) of the outcomes of all policies of \( P_1 \) against all policies of \( P_2 \). Thus, the following result follows from Theorem II.1.

Corollary III.2 (A-priori bounds) Assume that both players use the same set of heuristics \( \text{Heur}_1 \) and \( \text{Heur}_2 \) and same set of update functions \( \text{Update}_1 \) and \( \text{Update}_2 \) in Algorithm 4. If \( n_1 \) satisfies Eq. (1) for some \( \tilde{n}_2 \geq n_2 \), then Algorithm 3 is \( \epsilon \)-secure for \( P_1 \) with confidence \( 1 - \delta \). If one further increases \( n_1 \) to satisfy Eq. (2), then, with probability \( 2 \) higher than \( 1 - \beta \), the policy \( y^* \) with value \( V(A_1) \) is \( \epsilon \)-secure for \( P_1 \) with confidence \( 1 - \delta \).

2The confidence level \( 1 - \beta \) for \( P_1 \) refers solely to the generation of the matrix \( A_1 \) which defines \( y^* \) and \( V(A_1) \).

Akin to Section II, we now address a-posteriori guarantees that can be provided with a value of \( n_1 \) lower than that given by Corollary III.2, but with a higher value of \( \epsilon \). For brevity, we only consider the procedure, presented in Algorithm 5, from the perspective of \( P_1 \).

Steps 1 – 4 are identical to that of Algorithm 3. In step 5, the policy \( \bar{\sigma} \) is re-initialized, which ensures that the partially defined policies \( \gamma \) are played against \( k_1 \) initialized (new) policies of the opponent (Step 6). Note that the \( \gamma \)'s are not re-initialized, which ensures that the new \( k_1 \) policies of the opponent will be played against the partially defined policies of \( P_1 \) obtained from steps 1 – 4.

Algorithm 5: A-posteriori procedure

1. **P1 Initialization**: \( \gamma := [\gamma_1, \ldots, \gamma_{m_1}] = [\emptyset, \ldots, \emptyset] \), \( \bar{\sigma} := [\bar{\sigma}_1, \ldots, \bar{\sigma}_{n_1}] = [\emptyset, \ldots, \emptyset] \).

2. Generate sub-matrix: 
   \[ A_1, \gamma, \bar{\sigma} = \text{GenMatrix}(m_1, n_1, \gamma, \bar{\sigma}) \]

3. Security policy: \( y_1^* \in \arg\min_{y_1 \in S_{n_1}} \max_{z \in S_{m_1}} y_1' A_1 z \)

4. Security value: \( V(A_1) = \max_{y_1 \in S_{n_1}} y_1' A_1 z \)

5. **Re-Initialization**: \( \sigma := [\sigma_1, \ldots, \sigma_{k_1}] := [\emptyset, \ldots, \emptyset] \)

6. Generate sub-matrix: 
   \[ B, \gamma, \bar{\sigma} = \text{GenMatrix}(m_1, k_1, \gamma, \bar{\sigma}) \]

Output: A-posteriori security level: 
\[ \bar{v} := \max_{j \in \{1, \ldots, k_1\}} y_1^* B \]

Similar to Theorem II.2, we can show the following a-posteriori guarantee for dynamic games.

Corollary III.3 (A-posteriori bounds) Suppose that both players use the same set of heuristics \( \text{Heur}_1 \) and \( \text{Heur}_2 \) and same set of update functions \( \text{Update}_1 \) and \( \text{Update}_2 \) in Algorithm 4. Then, with \( k_1 \) satisfying Eq. (3) for some \( \tilde{n}_2 \geq n_2 \), the Algorithm 3 is \( \epsilon \)-secure for \( P_1 \) with confidence \( 1 - \delta \) for any \( \epsilon \) satisfying Eq. (4). If one further increases \( k_1 \) to satisfy Eq. (5), then, with probability \( \epsilon \) higher than \( 1 - \beta \), the policy \( y^* \) with value \( V(A_1) \) is \( \epsilon \)-secure for \( P_1 \) with confidence \( 1 - \delta \).

IV. Dynamic games with Dominating Heuristics

Suppose that \( P_1 \) is aware of a good heuristic that \( P_2 \) may apply to play the game. Then \( P_1 \) should secure herself by playing against those better heuristics of \( P_2 \) while constructing her sub-matrix \( A_1 \). This motivates the scenario of policy domination, which we discuss in this section.

A. Review: Matrix games with Dominated strategies

Definition 1 (\( \epsilon \)-Dominance in Matrix Games) Given an \( M \times N \) matrix \( A \) and two left-stochastic (0,1)-matrices \( \Pi^* \in \mathbb{B}^{N \times n^*} \) and \( \Pi \in \mathbb{B}^{N \times n} \), \( \Pi^* \) is said to \( \epsilon \)-dominate \( \Pi \)

3The confidence level \( 1 - \beta \) for \( P_1 \) refers solely to the generation of the matrix \( A_1 \) which defines \( y^* \) and \( V(A_1) \).
for some $\epsilon \geq 0$ if there exists a policy $z_{\text{dom}} \in S_n^*$ such that, for every $j \in \{1, \ldots, n\}$ and for every $i \in \{1, \ldots, M\}$,
$$e_i(M)'A\pi^*z_{\text{dom}} \leq e_i(M)'A\pi e_j(n) + \epsilon.$$  

For instance, if $\epsilon = 0$, then the column choice of $\pi$ is worse for $P_1$ and better for $P_2$ than the columns $\Pi^*$. The next definition essentially states that in such a case, the probability of sampling a column from $\Pi$ is worse for $P_1$ and better for $P_2$, should be higher for $P_1$ than that for $P_2$.

**Definition 2 (Perturbed sampling)** Let $\Pi^* \in B^{N \times n^*}$ and $\Pi \in B^{N \times n}$ two left-stochastic $(0,1)$-matrices. Given two probability measures $P_{\Delta} (\cdot)$ and $P_{\Delta}^* (\cdot)$ on $B^{N \times n}$, $P_{\Delta}^* (\cdot)$ is a perturbation of $P_{\Delta} (\cdot)$ with respect to the pair $(\Pi, \Pi^*)$ if

1) the probability of selecting a column which is neither in $\Pi$ nor in $\Pi^*$ is equal in both measures, i.e.,
$$P_{\Delta} (e_j(N) \in \text{Range}(\Delta)) = P_{\Delta}^* (e_j(N) \in \text{Range}(\Delta^*)), $$
for all $j$ such that $e_j(N) \notin \text{Range}(\Pi^*) + \text{Range}(\Pi)$, where the operation $+$ denotes sub-space sum.

2) the probability of selecting a column which is in $\Pi$ is larger for the measure $P_{\Delta}^* (\cdot)$, i.e.,
$$P_{\Delta} (e_j(N) \in \text{Range}(\Delta)) \leq P_{\Delta}^* (e_j(N) \in \text{Range}(\Delta^*)), $$
for all $j$ such that $e_j(N) \in \text{Range}(\Pi)$.

Then, the following result holds for $P_1$.

**Theorem IV.1 (Domination)** Given the matrix $A$, suppose that for some $\epsilon \geq 0$, there exists a matrix $\Pi^* \in B^{N \times n^*}$ which $\epsilon$-dominates another matrix $\Pi \in B^{N \times n}$. If the probability distributions of $\Pi_1$ and $\Pi_2$ are $\epsilon$-perturbed, then, with $n_1$ satisfying Eq. (1), for some $\tilde{n}_2 \geq n_2$, the SSP algorithm is $\epsilon$-secure for $P_1$ with confidence $1 - \delta$. If one further increases $n_1$ to satisfy Eq. (2), for some $\beta \in (0, 1)$, then, with probability higher than $1 - \beta$, the policy $y^*$ with value $V(A_1)$ is $\epsilon$-secure for $P_1$ with confidence $1 - \delta$.

**B. Extension to Dynamic Games**

The sub-matrix $A_1$, and therefore, its entries $a_{ij}$ generated by Algorithm 4 can be viewed as functions of the stochastic heuristics $\text{Heur}_1$ and $\text{Heur}_2$ used by the players. Now, suppose that for some $\epsilon \geq 0$, the heuristic $\text{Heur}_2$ $\epsilon$-dominates another heuristic $\overline{\text{Heur}}_2$ in the sense that for each information state $I$, for every outcome $\omega \in \Omega$, and for every $(i, j)$,
$$a_{ij}(\text{Heur}_1, \text{Heur}_2(I, \omega)) \leq a_{ij}(\overline{\text{Heur}}_1, \overline{\text{Heur}}_2(I, \omega)) + \epsilon.$$  

If $\epsilon = 0$, then $\overline{\text{Heur}}_2$ is a superior heuristic for $P_2$ as compared to $\text{Heur}_2$, and so, $P_1$ might as well assume that $P_2$ will use $\overline{\text{Heur}}_2$ instead of $\text{Heur}_2$ to avoid an unpleasant surprise. This is formalized by the next result, which essentially extends Theorem IV.1 to dynamic games.

**Proposition IV.2 (Dominating Heuristics)** Suppose that $P_1$ uses $\overline{\text{Heur}}_2$ instead of $\text{Heur}_2$, which satisfies Eq. (6) for some $\epsilon \geq 0$. Then, with $n_1$ satisfying Eq. (1), for some $\tilde{n}_2 \geq n_2$, the Algorithm 3 is $\epsilon$-secure for $P_1$ with confidence $1 - \delta$. If one further increases $n_1$ to satisfy Eq. (2), then, with probability higher than $1 - \beta$, the policy $y^*$ with value $V(A_1)$ is $\epsilon$-secure for $P_1$ with confidence $1 - \delta$.

**Proof:** Consider the large $M \times N$ matrix $A$ obtained by playing every completely defined policy of $P_1$ against that of $P_2$ by using $\text{Heur}_2$. Similarly, let $\tilde{A}$ denote the matrix thus obtained by using $\overline{\text{Heur}}_2$ instead of $\text{Heur}_2$. If we now consider the concatenated matrix $\tilde{A} := [A \tilde{A}]$, then Algorithm 3 is equivalent to using Algorithm 1 on the matrix $\tilde{A}$, from which $P_1$ and $P_2$ sample columns of their sub-matrices using the measures $P_{\Pi_1} (\cdot)$ and $P_{\Pi_2} (\cdot)$, respectively, where $\Pi_1 \in B^{N \times n_1}$ and $\Pi_2 \in B^{N \times n_2}$.

Since the heuristics $\text{Heur}_2$ and $\overline{\text{Heur}}_2$ satisfy Eq. (6), there exists a matrix $\Pi^* \in B^{2N \times M}$ (corresponding to the columns of $A$) which $\epsilon$-dominates another matrix $\Pi \in B^{2N \times M}$ (corresponding to the columns of $\tilde{A}$) with respect to the matrix $\tilde{A}$. Further, when $P_1$ uses $\text{Heur}_2$ in place of $\overline{\text{Heur}}_2$, the measures $P_{\Pi_1} (\cdot)$ and $P_{\Pi_2} (\cdot)$ are perturbed with respect to the pair $(\Pi, \Pi^*)$. The result now follows by applying Theorem IV.1 to the matrix $\tilde{A}$.

V. EXAMPLE: PARTIAL INFORMATION TIC-TAC-TOE

We now apply the SSP procedure from Section III to a partial-information version of the Tic-Tac-Toe game. Partial information means that neither player can see the entries in certain squares. The rules of this game are as follows.

1) Player $P_1$ gets to play first. The game proceeds with alternate moves. $P_1$ marks squares with X’s and $P_2$ with O’s. Neither player can see the entries in squares 2, 6 and 7 of the board, as shown in Figure 1.

2) If a player plays into one of the squares 2, 6 or 7, and the opponent has already played into that square, then the player misses the turn, and the mark that was already in the square stays. However, the player is not informed of this occurrence.

3) The game terminates when there are either three X’s or three O’s in a row or column or diagonal. The game ends in a draw if all the squares of the board are full and neither player has won. The value of the game is $-1, +1$, or 0 depending on whether $P_1$ wins, loses or the game ends in a draw respectively.

We implemented Algorithm 3 with identical heuristics for Algorithm 4, i.e., $\text{Heur}_1 = \text{Heur}_2 = \text{Heur}$, described below.

![Fig. 1. Partial Information Tic-Tac-Toe. Neither player can view the entries in the shaded squares. The first player to play into a shaded square gets the square. The goal is to get three squares in a row or column or a diagonal.](image_url)
from the point of view of $P_1$. Let $\hat{I}(I)$, initialized to $\emptyset$, denote an estimate of the true state of the board.

1) If a winning move can be deduced from the visible moves, play the winning move. Thus, given the estimate $I$, if $A_{win}$ is the set of winning moves, then $\text{Heur}(\hat{I}(I), \omega) \in A_{win}$.

2) Otherwise, if $P_2$ can win the game in the next move, then block $P_2$’s move. Thus, if $A_{\text{block}}$ denotes the set of moves that prevent $P_2$ from winning, then $\text{Heur}(\hat{I}(I), \omega) \in A_{\text{block}}$.

3) Otherwise, uniformly randomly select a square to play into from the allowed squares. Thus, $\text{Heur}(\hat{I}(I), \omega) \in \text{rand}(A)$.

4) Update $\hat{I}(I)$ after every move. If $P_1$ believes that she should have won the game, and yet the game has not ended, then toggle the entry (change $X$ to $O$) of the hidden square in the corresponding row/column/diagonal in her estimate of the board. The functions $\text{Update}_{1}$, $\text{Update}_{2}$ update the information state $I$ if the move of a player is visible; and leave $I$ unchanged if the move of a player is in the hidden.

For a fixed value of $\bar{n}_2$, $\beta$, and $\delta$, the a-posteriori security level $\bar{\nu}$ is determined by testing the sampled security policy $y^*$ against $k_1$ new policies of the opponent, where $k_1$ is determined by bounds in Corollary III.3. To generate the plots in Figure 2, we ran several Monte Carlo simulations of the SSP algorithm for the partial information Tic-Tac-Toe game. Since $\bar{\nu}$ is a random variable, it takes different values in the different Monte Carlo simulations. Figure 2 shows the solid 90 (resp. dashed 50) percentile curve such that 90% (resp. 50%) of the realizations of $\bar{\nu}$ were below this curve.

We observe that both the 90 and the 50 percentile curves for $\bar{\nu}$ are relatively “flat”, implying that with a much lesser number of columns $n_1$ (five times lesser), $P_1$ obtains only a small increase (by 15 per cent) in the 90-percentile value of a-posteriori security level. Thus, by using a much smaller number of column samples, significant computational savings can be obtained at the expense of a relatively small increase in the a-posteriori security level.

VI. CONCLUSION AND FUTURE DIRECTIONS

We addressed the use of sampling methods to provide probabilistic guarantees of security in partial information dynamic games. We presented a general procedure that allows for the solution of partial information dynamic games. We provided bounds on the sizes of the sub-matrices to be generated, which guarantee that with a sufficiently high probability, the outcome of the dynamic game will not violate the anticipated value of the game. Finally, we demonstrated the effectiveness of our procedure on a partial information version of the classic Tic-Tac-Toe game, for which no deterministic security levels or strategies have been published.

Extensions of this work to games with continuous decision spaces is an important future direction. Challenging problems also include providing probabilistic guarantees when players do not possess perfect recall of all past actions.

REFERENCES