Tree-Based Stochastic Optimization for Solving Large-Scale Urban Network Security Games

Shuxin Zhuang[1,2*], **Linjian Meng**[3*], **Shuxin Li**[4], **Minming Li**[1†], **Youzhi Zhang**[2†]

[1]Department of Computer Science, City University of Hong Kong, HKSAR, China
[2]CAIR, Hong Kong Institute of Science & Innovation, CAS, HKSAR, China
[3]National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
[4]College of Computing and Data Science, Nanyang Technological University, Singapore
shuxin.zhuang@my.cityu.edu.hk; menglinjian@smail.nju.edu.cn; shuxin.li@ntu.edu.sg;
minming.li@cityu.edu.hk; youzhi.zhang@cair.cas.org.hk

Abstract

Urban Network Security Games (UNSGs), which model the strategic allocation of limited security resources on city road networks, are critical for urban safety. However, finding a Nash Equilibrium (NE) in large-scale UNSGs is challenging due to their massive and combinatorial action spaces. One common approach to addressing these games is the Policy-Space Response Oracle (PSRO) framework, which requires computing best responses (BR) at each iteration. However, precisely computing exact BRs is impractical in large-scale games, and employing reinforcement learning to approximate BRs inevitably introduces errors that limit the overall effectiveness of the PSRO methods. Recent advancements in leveraging non-convex stochastic optimization to approximate an NE offer a promising alternative to the burdensome BR computation. However, utilizing existing stochastic optimization techniques with an unbiased loss function for UNSGs remains challenging because the action spaces are too vast to be effectively represented by neural networks. To address these issues, we introduce **T**ree-based **S**tochastic **O**ptimization (TSO), a framework that bridges the gap between the stochastic optimization paradigm for NE-finding and the demands of UNSGs. Specifically, we employ the tree-based action representation that maps the whole action space onto a tree structure, addressing the challenge faced by neural networks in representing actions when the action space cannot be enumerated. We then incorporate this representation into the loss function and theoretically demonstrate its equivalence to the unbiased loss function. To further enhance the quality of the converged solution, we introduce a sample-and-prune mechanism that reduces the risk of being trapped in suboptimal local optima. Extensive experimental results indicate the superiority of TSO over other baseline algorithms in addressing the UNSGs.

Code — https://github.com/sxzhuang/TSO_UNSG
Extended version — https://arxiv.org/pdf/2511.10072

1 Introduction

The strategic allocation of limited security resources is crucial for public safety, with applications focused on preventing, deterring, and detecting illicit activities to enhance urban security (Tsai et al. 2010; Jain et al. 2011; Jain, Conitzer, and Tambe 2013; Iwashita et al. 2016; Zhang et al. 2017, 2019; Li et al. 2024). Many real-world security challenges, including the protection of critical infrastructure (Jain et al. 2011) and the interdiction of urban criminals (Zhang et al. 2017, 2019; Xue et al. 2021), can be effectively modeled as UNSGs. In these games, a team of defenders strategically deploys resources, such as police patrols or security checkpoints, across a city's road network to interdict the adversary. The adversary, in turn, aims to traverse the network from a source to a destination while minimizing the probability of being caught. However, a fundamental challenge in solving UNSGs lies in their computational complexity. As urban networks scale, the exponential growth of possible adversary paths and defender resource allocations leads to massive action spaces (Jain et al. 2011), making exact solutions computationally intractable.

A prominent line of research for solving UNSGs relies on iterative and oracle-based algorithms to approximate a Nash equilibrium (NE) (Nash 1951; McMahan, Gordon, and Blum 2003; Jain et al. 2011). A foundational method in this domain is the double-oracle algorithm (McMahan, Gordon, and Blum 2003; Jain et al. 2011; Zhang et al. 2017, 2019). It begins with a small, restricted set of pure strategies for each player. In each iteration, it computes an equilibrium for this restricted game and then expands the strategy sets by adding each player's best response (BR) to the opponent's current strategy. This process continues until convergence. While the classic double-oracle algorithm relies on linear programming, it loses its effectiveness in large-scale UNSGs. To overcome this limitation, PSRO (Lanctot et al. 2017) was introduced as a generalization of the double-oracle framework, embedding deep reinforcement learning (DRL) to handle immense strategy spaces, and it inherits the convergence guarantees of the double-oracle method. However, a critical bottleneck shared by these approaches is their dependence on best-response computation. Finding a best response in UNSGs is an NP-hard problem (Jain et al. 2011). Indeed, recent work (Xue et al. 2021; Xue, An, and Yeo 2022; Li et al. 2023, 2024; Zhuang et al. 2025) employs DRL to approximate the BR oracle for solving UNSGs, but these RL-based oracles often suffer from low accuracy. The resulting imprecise best

[*]These authors contributed equally.
[†]Corresponding authors.

responses hamper the effectiveness of the PSRO framework, and lead to convergence towards suboptimal policies.

In parallel, another line of research (Gemp et al. 2022; Gemp, Marris, and Piliouras 2024; Meng et al. 2025), primarily developed for normal-form games, reframes the NE-finding problem as a non-convex stochastic optimization task. In this paradigm, a loss function is constructed whose minimization corresponds to finding an NE. Player policies, often parameterized as neural networks, are then updated directly via gradient descent using batches of data from sampled gameplay, thereby bypassing the need for an explicit BR oracle. While this paradigm is promising, its direct application to large-scale UNSGs remains challenging. Its core mechanism requires representing a player's policy as an explicit probability distribution over the entire action space to facilitate action sampling. Such an enumeration-based representation becomes intractable in typical UNSGs, where the action space is vast and combinatorial.

In this paper, we propose a novel framework, Tree-based Stochastic Optimization (TSO), to solve large-scale UNSGs. Our approach bridges the gap between the stochastic optimization paradigm for NE-finding and the practical demands of large-scale security games. First, we introduce a tree-based action sampling process where each action is mapped to a unique path in a decision tree. This structure decomposes the probability of selecting an action into a product of conditional probabilities at sequential decision points, thereby enabling efficient action sampling without enumerating the full action space. We then incorporate the tree-based action representation into the Nash Advantage Loss (NAL) (Meng et al. 2025) and theoretically demonstrate its equivalence to the unbiased loss function. Furthermore, to enhance the quality of the converged solution, we introduce a sample-and-prune mechanism. This technique diversifies exploration by pruning the initially sampled high-probability action and forcing a re-sample from the remaining action space, thereby mitigating the risk of premature convergence to a suboptimal local optimum. Extensive experiments demonstrate that TSO significantly outperforms established baselines in UNSGs with both small and large action spaces.

2 Preliminaries

2.1 Game Definition

UNSGs involve a defender strategically placing security resources on city roads to protect against an adversary who selects a path through the city. Following the game definition from (Jain et al. 2011; Tsai et al. 2010), we model the UNSGs on a graph $G = (V, E)$ as a one-shot and simultaneous-move game between an attacker and a team of N defenders. The attacker's action, a_{attacker}, is a simple path from a start vertex $s \in V_{\text{start}}$ to a target vertex $t \in V_{\text{target}}$, with the action space $\mathcal{A}_{\text{attacker}}$ being the set of all such paths. Each defender $m \in \{1, \dots, N\}$ selects an edge e_m from their individual action space $\mathcal{E}_m \subseteq E$. The team's action a_{defender} is a tuple of these selected edges, and their action space is the Cartesian product $\mathcal{A}_{\text{defender}} = \times_{m=1}^{N} \mathcal{E}_m$.

Player utilities are determined by their joint actions. A function $U : V_{\text{target}} \to \mathbb{R}^+$ assigns a positive value $U(v_{t_i})$ to each target v_{t_i}, and we use $\text{target}(a_{\text{attacker}})$ to denote the endpoint of the attacker's chosen path. Interception occurs if the attacker's path a_{attacker} includes any edge from the defenders' action a_{defender}. A successful attacker gains utility equal to the target's value, $U(\text{target}(a_{\text{attacker}}))$, while an intercepted attacker incurs a penalty of the same magnitude. Formally, the attacker's utility is defined as:

$$u_{\text{attacker}}(a_{\text{attacker}}, a_{\text{defender}}) = \begin{cases} U(\text{target}(a_{\text{attacker}})) & \text{if } a_{\text{attacker}} \cap a_{\text{defender}} = \emptyset \\ -U(\text{target}(a_{\text{attacker}})) & \text{if } a_{\text{attacker}} \cap a_{\text{defender}} \neq \emptyset. \end{cases}$$

This game can be modeled as a zero-sum normal-form game (NFG), where the defender's utility is the negative of the attacker's utility: $u_{\text{defender}}(a_{\text{attacker}}, a_{\text{defender}}) = -u_{\text{attacker}}(a_{\text{attacker}}, a_{\text{defender}})$.

A mixed strategy for a player $i \in \mathcal{P}$, where $\mathcal{P} = \{\text{attacker}, \text{defender}\}$, denoted by $\boldsymbol{x}_i$, is a probability distribution over their pure action set $\mathcal{A}_i$. The space of all valid mixed strategies for player i is the probability simplex $\boldsymbol{\mathcal{X}}_i = \{\boldsymbol{x}_i \in \mathbb{R}^{|\mathcal{A}_i|} \mid \sum_{a_i \in \mathcal{A}_i} x_i(a_i) = 1, \forall a_i, x_i(a_i) \geq 0\}$. A strategy profile $\boldsymbol{x} = (\boldsymbol{x}_i)_{i \in \mathcal{P}}$ for the game is an element of the joint strategy space $\boldsymbol{\mathcal{X}} = \times_{i \in \mathcal{P}} \boldsymbol{\mathcal{X}}_i$. We denote the interior of this space as $\boldsymbol{\mathcal{X}}^\circ$, where every pure strategy is played with a strictly positive probability. Given a strategy profile $\boldsymbol{x}$, the expected utility for player i is given by $u_i(\boldsymbol{x}) = u_i(\boldsymbol{x}_i, \boldsymbol{x}_{-i})$, where $\boldsymbol{x}_{-i}$ denotes the strategies of all players except i. The expected utility for player i is:

$$u_i(\boldsymbol{x}_i, \boldsymbol{x}_{-i}) = \sum_{\boldsymbol{a} \in \times_{i \in \mathcal{P}} \mathcal{A}_i} u_i(\boldsymbol{a}) \prod_{j \in \mathcal{P}} \boldsymbol{x}_j(a_j).$$

Our primary objective is to compute an NE, a strategy profile $\boldsymbol{x}^*$ from which no player has a unilateral incentive to deviate. As analyzed in (Facchinei and Pang 2003), if the utility function of each player i is concave over $\boldsymbol{\mathcal{X}}_i$, an NE $\boldsymbol{x}$ is such that $\langle \nabla_{x_i} u_i(\boldsymbol{x}), \boldsymbol{x}_i - \boldsymbol{x}'_i \rangle \leq 0, \ \forall i \in \mathcal{P}$ and $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$. For NFGs, the expected utility function $u_i(\boldsymbol{x})$ is linear with respect to a player's own strategy $\boldsymbol{x}_i$. This property allows us to quantify how close a strategy profile $\boldsymbol{x}$ is to an equilibrium using the duality gap, which measures the total exploitability of the profile:

$$\text{dg}(\boldsymbol{x}) = \sum_{i \in \mathcal{P}} \left(\max_{\boldsymbol{x}'_i \in \boldsymbol{\mathcal{X}}_i} u_i(\boldsymbol{x}'_i, \boldsymbol{x}_{-i}) - u_i(\boldsymbol{x}_i, \boldsymbol{x}_{-i}) \right).$$

A strategy profile $\boldsymbol{x}$ is an NE if and only if its duality gap is zero. Our goal is to find a strategy profile $\boldsymbol{x}^* \in \boldsymbol{\mathcal{X}}$ such that $\text{dg}(\boldsymbol{x}^*) = 0$. A table of symbols is provided in Appendix F.

2.2 Nash Equilibrium as Stochastic Optimization

Many studies (Raghunathan, Cherian, and Jha 2019; Goktas et al. 2022; Marris et al. 2022; Gemp et al. 2022; Duan et al. 2023; Liu et al. 2024; Yongacoglu et al. 2024) treat computing NE as an optimization task. While various loss functions have been proposed for this purpose, many are not amenable to unbiased estimation. Recently, Gemp, Marris, and Piliouras (2024) and Meng et al. (2025) introduced methods based on unbiased estimation.

Gemp, Marris, and Piliouras (2024) leverage a property of concave games: for an interior strategy profile $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}^\circ$, a

player's utility gradients are identical across all actions if and only if $\boldsymbol{x}$ is an NE. To guarantee the existence of an interior NE, they introduce an entropy-regularized utility function for each player i: $u_i^\tau(\boldsymbol{x}) = u_i(\boldsymbol{x}) - \tau \boldsymbol{x}_i^\top \log \boldsymbol{x}_i$, where $\tau > 0$. Based on this insight, they proposed the first unbiased loss function $\mathcal{L}_G^\tau(\boldsymbol{x})$ for stochastic optimization.

However, as Meng et al. (2025) point out, $\mathcal{L}_G^\tau(\boldsymbol{x})$ suffers from high-variance loss estimates, which can impede convergence. To address this, Meng et al. (2025) introduced the surrogate Nash Advantage Loss (NAL), $\mathcal{L}_{\text{NAL}}^\tau(\boldsymbol{x})$, designed to enable low-variance, unbiased estimation using only a single random variable. The loss is defined as:

$$\mathcal{L}_{\text{NAL}}^\tau(\boldsymbol{x}) = \sum_{i \in \mathcal{P}} \langle \text{sg}[\boldsymbol{F}_i^{\tau,\boldsymbol{x}} - \langle \boldsymbol{F}_i^{\tau,\boldsymbol{x}}, \hat{\boldsymbol{x}}_i \rangle \mathbf{1}], \boldsymbol{x}_i \rangle, \tag{1}$$

where $\boldsymbol{F}_i^{\tau,\boldsymbol{x}} = -\nabla_{\boldsymbol{x}_i} u_i^\tau(\boldsymbol{x})$, $\hat{\boldsymbol{x}} \in \boldsymbol{\mathcal{X}}$ can be any strategy profile (with $\hat{\boldsymbol{x}}_i \neq \mathbf{0}$ for all $i \in \mathcal{P}$), and $\text{sg}[\cdot]$ is the stop-gradient operator that implies the term in this operator is not involved in gradient backpropagation. So the first-order gradient of NAL is:

$$\begin{aligned} \nabla_{\boldsymbol{x}_i} \mathcal{L}_{\text{NAL}}^\tau(\boldsymbol{x}) &= \text{sg}\left[\boldsymbol{F}_i^{\tau,\boldsymbol{x}} - \langle \boldsymbol{F}_i^{\tau,\boldsymbol{x}}, \hat{\boldsymbol{x}}_i \rangle \mathbf{1}\right] \\ &= -\nabla_{\boldsymbol{x}_i} u_i^\tau(\boldsymbol{x}) + \langle \nabla_{\boldsymbol{x}_i} u_i^\tau(\boldsymbol{x}), \hat{\boldsymbol{x}}_i \rangle \mathbf{1}. \end{aligned}$$

Specifically, $\nabla_{\boldsymbol{x}_i} \mathcal{L}_{\text{NAL}}^\tau(\boldsymbol{x}) = \mathbf{0}$ if and only if the gradients for all actions are identical, which is equivalent to the global minimum of $\mathcal{L}_G^\tau(\boldsymbol{x})$. Therefore, NAL employs a single random variable to obtain an unbiased estimate of the loss function, which reduces variance and transforms the problem of finding an NE into a stochastic optimization task.

3 Methodology

In this section, we introduce the **T**ree-based **S**tochastic **O**ptimization (TSO) framework. First, we address the limitation of neural networks in representing non-enumerable actions by introducing the tree-based action representation. We then integrate this representation into the NAL and prove its equivalence. Additionally, we propose a sample-and-prune mechanism to mitigate the risk of converging to local optima.

3.1 Tree-Based Action Representation

Using a tree to model each player's action space avoids the need to enumerate all possible actions. In this approach, actions are generated through a sequence of decisions at the nodes of the tree. Since the attacker and defender use different processes to generate actions, we propose two different tree construction strategies, each specifically designed for their respective action spaces.

Tree Construction of Attacker Action. The core idea of constructing a tree to represent the attacker's action space is to map each simple path in the graph $G = (V, E)$ to a unique path in the tree. We construct a tree, denoted as $\mathcal{T}_{\text{attacker}}$, which decomposes an attacker action into a series of steps, with each step corresponding to a node in $\mathcal{T}_{\text{attacker}}$. An example illustrating the construction of the attacker's action representation tree is shown in Figure 1. The tree is constructed according to the following principles:

- **Tree Structure and Nodes:** Each node $z \in \mathcal{T}_{\text{attacker}}$ represents a state in the attacker's path-finding process and is defined by two components:

  1. Action Component (a_z^{att}): For any non-root node z, $a_z^{att} \in V$ is the vertex visited to transition from its parent to z. For the root node, z_{root}, which represents the state before any action, $a_{z_{\text{root}}}^{att} = \emptyset$.

  2. History Component (h_z^{att}): This component records the sequence of actions taken from z_{root} to a node z, providing the information for subsequent decisions. Formally, for a path of nodes $(z_0, z_1, \ldots, z_i)$ from the root ($z_0 = z_{\text{root}}$) to the current node z_i, its history is the corresponding sequence of actions: $h_{z_i}^{att} = (a_{z_1}^{att}, a_{z_2}^{att}, \ldots, a_{z_i}^{att})$. The history of the root node is empty: $h_{z_{\text{root}}}^{att} = \emptyset$.
- **Children Construction:** The children of any node $z \in \mathcal{T}_{\text{attacker}}$ are determined by the valid subsequent actions from the state represented by z. For the root node z_{root}, its children correspond to all possible starting vertices in V_{start}. For each $v \in V_{\text{start}}$, a child node z' is created with $a_{z'}^{att} = v$ and $h_{z'}^{att} = (v)$. For any non-root, non-leaf node z, the set of available next vertices is the set of neighbors of a_z^{att} in the graph G, denoted $N(a_z^{att})$. To maintain the simple path constraint and ensure the path can reach a target, the set of valid next actions is filtered. Specifically, a neighbor v' is considered valid if it has not been visited (i.e., $v' \notin h_z^{att}$) and there exists a simple path to a target via v'. The set of valid actions is denoted as $A_{\text{valid}}(z)$. For each vertex $v' \in A_{\text{valid}}(z)$, a child node z_{child} is generated, with its components defined as $a_{z_{\text{child}}}^{att} = v'$ and $h_{z_{\text{child}}}^{att} = h_z^{att} \oplus (v')$, where $\oplus$ denotes sequence concatenation.

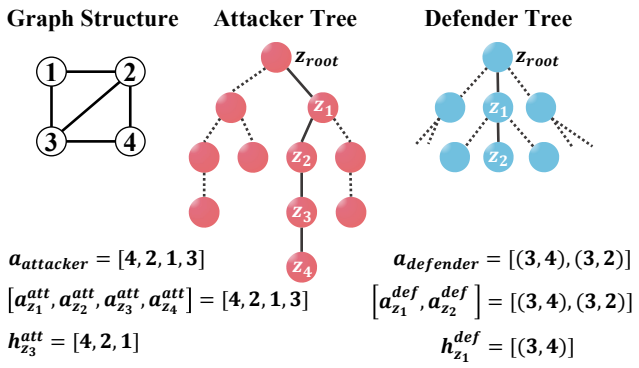

Figure 1: Construction of action representation trees. Vertices 2 and 4 are attacker starts and node 3 the target; defenders choose $(1, 3), (3, 2), (3, 4)$; attacker action [4,2,1,3] maps to the solid path $z_{\text{root}} \to z_4$ in the attacker tree, while defender action [(3,4), (3,2)] maps to $z_{\text{root}} \to z_2$ in the defender tree.

Path-to-Action Mapping. This construction ensures that any path from z_{root} to a leaf node z corresponds to a unique attacker action a_{attacker}. Therefore, the process of selecting an attacker's action reduces to applying a policy to navigate from the root to a leaf in this tree. Thus, there is a one-to-one mapping between all simple paths in G (from a vertex in V_{start} to a vertex in V_{target}) and the root-to-leaf paths in $\mathcal{T}_{\text{attacker}}$.

Action Probability. Based on the sequential decision-making process defined by $\mathcal{T}_{\text{attacker}}$, we now formalize the

probability of an attacker's action. An attacker's action, a_{attacker}, is a simple path from a starting vertex to a target, represented by $a_{\text{attacker}} = (v_1, v_2, \ldots, v_L)$. This sequence corresponds to a unique root-to-leaf path $(z_0, z_1, \ldots, z_L)$ in $\mathcal{T}_{\text{attacker}}$, where $z_0 = z_{\text{root}}$ and for each step $i \in \{1, \ldots, L\}$, node z_i is chosen from the children of z_{i-1} with its action component being $a^{att}_{z_i} = v_i$. The selection of each vertex in the sequence is controlled by a policy, π_θ, parameterized by θ. At each node z_{i-1} on the path, the policy computes a probability distribution over all valid next actions. The probability of selecting the specific next vertex v_i (and thus transitioning to node z_i) is conditioned on the $h^{att}_{z_{i-1}}$. This conditional probability is denoted as $\pi_\theta(a^{att}_{z_i} \mid h^{att}_{z_{i-1}})$. Applying the chain rule of probability, the likelihood of a_{attacker} is the product of the conditional probabilities for each decision along the corresponding path in $\mathcal{T}_{\text{attacker}}$: $\pi_\theta(a_{\text{attacker}}) = \prod_{i=1}^{L} \pi_\theta(a^{att}_{z_i} \mid h^{att}_{z_{i-1}})$

Tree Construction of Defender Action. The construction of the action representation tree for the defense team is similar to that of the attacker. Each node z in the $\mathcal{T}_{\text{defender}}$ consists of two components: the Action Component (a^{def}_z) and the History Component (h^{def}_z). The primary difference is that, in $\mathcal{T}_{\text{defender}}$, a^{def}_z represents an edge in the graph, while h^{def}_z denotes a sequence of edges. Further details regarding the construction of $\mathcal{T}_{\text{defender}}$ are provided in Appendix A.1.

Advantages. A key advantage of our tree-based action representation is that neither $\mathcal{T}_{\text{attacker}}$ nor $\mathcal{T}_{\text{defender}}$ needs to be constructed in advance. Instead, we employ neural networks to represent the tree structure. Child nodes are generated dynamically at each node z, where the neural network takes the history component h_z as input and makes decisions based on the historical information. Sampling only considers current-level candidates, yielding $O(d|E|)$ time per sampled action (with d the maximum out-degree, $|E|$ the simple-path length bound).The output represents the probabilities of selecting each child node. Additionally, we employ action masking to handle the variable number of child nodes. Further implementation details are in Appendix A.2. As a result, our method eliminates the need to enumerate the entire action space.

3.2 Equivalence to NAL

This section formally shows how our tree-based action representation preserves the NAL property. Specifically, we demonstrate the equivalence between the tree-based and the original NAL. This ensures that optimizing the tree-based NAL shares the same objective as the original and thus satisfies the same NE conditions.

Notation. To facilitate the proof, we first establish our notation. For each player $i \in \mathcal{P}$, the set of available actions is $\mathcal{A}_i$. We use an index $k \in \{0, 1, \ldots, |\mathcal{A}_i| - 1\}$ to refer to a specific action in this set. A mixed strategy for player i is a probability distribution over $\mathcal{A}_i$, represented by the vector $\boldsymbol{x}_i = [\sigma_0, \sigma_1, \ldots, \sigma_{|\mathcal{A}_i|-1}]$, where σ_k is the probability of selecting the action indexed by k. We also consider another mixed strategy $\hat{\boldsymbol{x}}_i$ as used in the NAL.

In our method, the action probabilities σ_k are not atomic variables but are constructed via a tree-based action representation. Each player i has its own tree $\mathcal{T}_i$. Let E^i_{tree} be the set of all edges in this tree. For each edge $j \in E^i_{\text{tree}}$, we define $\dot{\sigma}_j$ as the probability of traversing that edge from its parent node to the child. Each player i's action k corresponds to a unique root-to-leaf path in $\mathcal{T}_i$. We denote the set of edges constituting this path as $S_k \subseteq E^i_{\text{tree}}$. The probability of player i selecting the action k is the product of the traversal probabilities of all edges along this path: $\sigma_k = \prod_{j \in S_k} \dot{\sigma}_j$.

Tree-Based NAL. Building upon the tree-based action representation and Eq.(1), we now derive the tree-based NAL as follows:

$$\begin{aligned}\mathcal{L}^{\tau}_{\text{TSO}}(\boldsymbol{x}) &= \sum_{i \in \mathcal{P}} \sum_{k=0}^{|\mathcal{A}_i|-1} \text{sg}[\boldsymbol{F}_i^{\tau,\boldsymbol{x}}(k) - \langle \boldsymbol{F}_i^{\tau,\boldsymbol{x}}, \hat{\boldsymbol{x}}_i \rangle] \sigma_k \\ &= \sum_{i \in \mathcal{P}} \sum_{k=0}^{|\mathcal{A}_i|-1} \text{sg}[\boldsymbol{F}_i^{\tau,\boldsymbol{x}}(k) - \langle \boldsymbol{F}_i^{\tau,\boldsymbol{x}}, \hat{\boldsymbol{x}}_i \rangle] \prod_{j \in S_k} \dot{\sigma}_j,\end{aligned}$$

where $\boldsymbol{F}_i^{\tau,\boldsymbol{x}}(k)$ denotes the value of $\boldsymbol{F}_i^{\tau,\boldsymbol{x}}$ corresponding to player i's k-th action; other terms are as in Eq.(1). Derivation details are provided in Appendix B.

First-Order Gradient of $\mathcal{L}^{\tau}_{\text{TSO}}(\boldsymbol{x})$. The optimization of $\mathcal{L}^{\tau}_{\text{TSO}}$ proceeds by updating the parameters of the policy, which are the edge probabilities $\{\dot{\sigma}_j\}_{j \in E^i_{\text{tree}}}$. To derive the gradients for these parameters, we focus the analysis on player i and consider the partial derivative of the loss with respect to one of its action probabilities σ_k, where $k \in \{0, 1, \ldots, |\mathcal{A}_i| - 1\}$. Due to the stop-gradient operator sg[$\cdot$], which treats its argument as a constant during differentiation, the derivative simplifies to:

$$\frac{\partial \mathcal{L}^{\tau}_{\text{TSO}}(\boldsymbol{x})}{\partial \sigma_k} = \text{sg}[\boldsymbol{F}_i^{\tau,\boldsymbol{x}}(k) - \langle \boldsymbol{F}_i^{\tau,\boldsymbol{x}}, \hat{\boldsymbol{x}}_i \rangle].$$

This term is precisely the first-order gradient for action k in the NAL. For notational clarity, we define:

$$g_{i,k} := \text{sg}[\boldsymbol{F}_i^{\tau,\boldsymbol{x}}(k) - \langle \boldsymbol{F}_i^{\tau,\boldsymbol{x}}, \hat{\boldsymbol{x}}_i \rangle].$$

Next, we use the multivariate chain rule to propagate this gradient back to an arbitrary edge probability $\dot{\sigma}_j$:

$$\frac{\partial \mathcal{L}^{\tau}_{\text{TSO}}(\boldsymbol{x})}{\partial \dot{\sigma}_j} = \sum_{k=0}^{|\mathcal{A}_i|-1} \frac{\partial \mathcal{L}^{\tau}_{\text{TSO}}(\boldsymbol{x})}{\partial \sigma_k} \frac{\partial \sigma_k}{\partial \dot{\sigma}_j} = \sum_{k=0}^{|\mathcal{A}_i|-1} g_{i,k} \frac{\partial \sigma_k}{\partial \dot{\sigma}_j}.$$

The partial derivative $\frac{\partial \sigma_k}{\partial \dot{\sigma}_j}$ is non-zero only if edge j is on the path S_k. If $j \in S_k$, this derivative is the product of all other edge probabilities on the path:

$$\frac{\partial \sigma_k}{\partial \dot{\sigma}_j} = \begin{cases} \prod_{l \in S_k, l \neq j} \dot{\sigma}_l & \text{if } j \in S_k \\ 0 & \text{otherwise.} \end{cases}$$

Let $\sigma_k^{\setminus j}$ denote $\prod_{l \in S_k, l \neq j} \dot{\sigma}_l$. The summation for the gradient thus reduces to only those actions whose paths include edge j. Let $\mathcal{A}_i(j) := \{k \in \{0, \ldots, |\mathcal{A}_i| - 1\} \mid j \in S_k\}$ be the set of indices for such actions. The final gradient expression is:

$$\frac{\partial \mathcal{L}^{\tau}_{\text{TSO}}(\boldsymbol{x})}{\partial \dot{\sigma}_j} = \sum_{k \in \mathcal{A}_i(j)} g_{i,k} \cdot \sigma_k^{\setminus j}.$$

Proposition 1. *For any edge j in $\mathcal{T}_i$, the first-order gradient of the tree-based NAL equals zero if and only if the first-order*

gradient of NAL is $\mathbf{0}$*, i.e.,*

$$\frac{\partial \mathcal{L}^{\tau}_{TSO}(\boldsymbol{x})}{\partial \dot{\sigma}_j} = 0 \quad \forall\, j \in E^i_{tree} \iff \boldsymbol{g}_i = \mathbf{0},$$

where $\boldsymbol{g}_i = [g_{i,0}, g_{i,1}, \ldots, g_{i,|\mathcal{A}_i|-1}]^\top$*. The detailed proof of this proposition is provided in Appendix C.*

According to Proposition 1, we demonstrate equivalence between $\mathcal{L}^{\tau}_{\text{TSO}}(\boldsymbol{x})$ and $\mathcal{L}^{\tau}_{\text{NAL}}(\boldsymbol{x})$ when their first-order derivatives are zero. This implies that optimization of $\mathcal{L}^{\tau}_{\text{TSO}}(\boldsymbol{x})$ is the same as optimization of $\mathcal{L}^{\tau}_{\text{NAL}}(\boldsymbol{x})$. Consequently, if the optimization of $\mathcal{L}^{\tau}_{\text{NAL}}(\boldsymbol{x})$ converges to an NE, it follows that the optimization of $\mathcal{L}^{\tau}_{\text{TSO}}(\boldsymbol{x})$ will converge to an NE.

Algorithm 1: TSO

1: **Input:** An optimizer $\mathcal{OPT}$, the player set $\mathcal{P} = \{\text{attacker}, \text{defender}\}$, the exploration ratio ϵ, the uniform strategy profile $\boldsymbol{x}^u = [\boldsymbol{x}^u_k | k \in \mathcal{P}]$, the initial parameter θ, the initial parameter ϕ, the learning rate η, the regularization scalar τ, the number of total iterations T, the number of instances S sampled per iteration, the frequency T_u of updating η and τ, the weight α on updating η, the weight β on updating τ, simulator $\mathcal{G}$ that returns player i's payoff given a joint action.
2: **for** each $t \in [1,\ 2, \cdots, T]$ **do**
3: $\quad \mathcal{M}_k \leftarrow \{\}, v_k \leftarrow 0,\ \forall k \in \mathcal{P}$
4: $\quad$ **for** each $s \in [1,\ 2, \cdots, S]$ **do**
5: $\quad\quad$ **// Our Sample-and-Prune Mechanism**
6: $\quad\quad a_k \sim \boldsymbol{x}^{\theta,\phi}_k,\ \forall k \in \mathcal{P}$
7: $\quad\quad \boldsymbol{a} \leftarrow (a_k)_{k \in \mathcal{P}}$
8: $\quad\quad \boldsymbol{x}'_k \leftarrow (1-\epsilon)\boldsymbol{x}^{\theta,\phi}_k + \epsilon \boldsymbol{x}^u_k,\ \forall k \in \mathcal{P}$
9: $\quad\quad \boldsymbol{x}'_k(a_k) \leftarrow 0,\ \forall k \in \mathcal{P}$
10: $\quad\quad \boldsymbol{x}'_k \leftarrow \frac{\boldsymbol{x}'_k}{\|\boldsymbol{x}'_k\|_1},\ \forall k \in \mathcal{P}$
11: $\quad\quad a_{k'} \sim \boldsymbol{x}'_k, p_k \leftarrow \boldsymbol{x}'_k(a_{k'}),\ \forall k \in \mathcal{P}$
12: $\quad\quad$ **// To estimate** $\boldsymbol{F}^{\tau,\boldsymbol{x}^{\theta,\phi}}_k(a_{k'})$
13: $\quad\quad r_k \leftarrow -\mathcal{G}(k, a_{k'}, \boldsymbol{a}_{-k}) + \tau \log \boldsymbol{x}^{\theta,\phi}_k(a_{k'}),\ \forall k \in \mathcal{P}$
14: $\quad\quad \mathcal{M}_k$.append$([i,\ a_{k'},\ r_k,\ p_k]),\ \forall k \in \mathcal{P}$
15: $\quad\quad v_k \leftarrow v_k + r_k$
16: $\quad$ **end for**
17: $\quad \hat{\mathcal{L}}^{\tau}_{\text{TSO}}(\theta, \phi) \leftarrow 0$
18: $\quad v_k \leftarrow \frac{v_k}{S},\ \forall k \in \mathcal{P}$ **// To estimate** $\langle \boldsymbol{F}^{\tau,\boldsymbol{x}^{\theta,\phi}}_k, \hat{\boldsymbol{x}}_k \rangle$
19: $\quad$ **for** each $k \in \mathcal{P}$ **do**
20: $\quad\quad$ **for** each $[i, a^s_k, r^s_k, p^s_k] \in \mathcal{M}_k$ **do**
21: $\quad\quad\quad$ **// To estimate** $\boldsymbol{F}^{\tau,\boldsymbol{x}^{\theta,\phi}}_k - \langle \boldsymbol{F}^{\tau,\boldsymbol{x}^{\theta,\phi}}_k, \hat{\boldsymbol{x}}_k \rangle \mathbf{1}$
22: $\quad\quad\quad \boldsymbol{g}^s_k \leftarrow \frac{r^s_k - v_k}{p^s_k} \boldsymbol{e}_{a^s_k}$
23: $\quad\quad\quad \hat{\mathcal{L}}^{\tau}_{\text{TSO}}(\theta, \phi) \leftarrow \hat{\mathcal{L}}^{\tau}_{\text{TSO}}(\theta, \phi) + \langle sg[\boldsymbol{g}^s_k], \boldsymbol{x}^{\theta,\phi}_k \rangle$
24: $\quad\quad$ **end for**
25: $\quad$ **end for**
26: $\quad \theta, \phi \leftarrow \mathcal{OPT}$.update$(\hat{\mathcal{L}}^{\tau}_{\text{TSO}}(\theta, \phi))$
27: $\quad$ **if** $t \% T_u = 0$ **then**
28: $\quad\quad \eta \leftarrow \alpha\eta, \tau \leftarrow \beta\tau$
29: $\quad$ **end if**
30: **end for**
31: **Return** θ, ϕ

3.3 Tree-Based Stochastic Optimization

In this section, we introduce the Sample-and-Prune mechanism and outline the TSO framework, showing how Tree-Based Action Representation and Sample-and-Prune are integrated to optimize our Tree-Based NAL.

Sample-and-Prune Mechanism. The core of optimizing the loss function with stochastic gradient descent lies in action sampling. We observe that, during the training process, a policy might discover a temporarily effective strategy and, through a strong positive feedback loop, quickly increase the probabilities of the related actions. This often results in convergence to a suboptimal local optimum. Consequently, the gradients computed from sampled actions approach zero, making it challenging for the agent to escape this suboptimal state, even when employing exploration methods such as ϵ-greedy with a high exploration rate.

To address this issue, we introduce the Sample-and-Prune mechanism. This approach decomposes the action sampling process into two distinct steps, with the final action being used to compute the loss and update the network parameters. The procedure is as follows:

1. **Sample:** An initial action, a_k, is sampled from the current policy distribution π.
2. **Prune and Re-sample:** The initially sampled action a_k is pruned from the set of available actions. A second action, $a_{k'}$, is then sampled from the same policy distribution π, conditioned on the pruned action space (i.e., $a_{k'} \neq a_k$).

Proposition 2. *The use of the Sample-and-Prune mechanism for action sampling does not affect the equivalence between* $\mathcal{L}^{\tau}_{TSO}(\boldsymbol{x})$ *and* $\mathcal{L}^{\tau}_{NAL}(\boldsymbol{x})$*.*

Further details on the sample-and-prune mechanism and the proof of Proposition 2 are provided in Appendix D.

Outline of TSO. Next, we present the pseudocode of TSO in Algorithm 1, illustrating the integration of Tree-Based Action Representation and the Sample-and-Prune Mechanism (Line 5-11) into stochastic optimization for optimizing the Tree-Based NAL. For clarity, we adopt the normal-form game strategy representation, denoting the strategy profile parameterized by θ and ϕ as $\boldsymbol{x}^{\theta,\phi} = (\boldsymbol{x}^{\theta,\phi}_{\text{attacker}}, \boldsymbol{x}^{\theta,\phi}_{\text{defender}})$, while omitting the details of our Tree-Based Action Representation.

Specifically, at each iteration t, for the s-th sample, the first step of our Sample-and-Prune Mechanism—the "Sample" phase—is to independently sample, for each player, an action a_k from the strategy profile $\boldsymbol{x}^{\theta,\phi}$ using our Tree-Based Action Representation (Line 6). This sampled a_k not only is used in our Sample-and-Prune Mechanism but also serves as an unbiased estimator for environmental dynamics, enabling unbiased estimation of $\mathcal{L}^{\tau}_{\text{TSO}}$. Formally, this sampled a_k for each $k \in \mathcal{P}$ yields the action profile $\boldsymbol{a} \leftarrow (a_k)_{k \in \mathcal{P}}$ (Line 7). For any player k, $\boldsymbol{a}_{-k} = (a_j)_{j \in \mathcal{P},\, j \neq k}$ serves as an unbiased estimate for environmental dynamics.

The second step is the "Prune and Re-sample" phase, which draws an alternative action $a_{k'} \neq a_k$ as follows: First, a modified strategy $\boldsymbol{x}'_k \leftarrow (1-\epsilon)\boldsymbol{x}^{\theta,\phi}_k + \epsilon \boldsymbol{x}^u_k$ is constructed for each $k \in \mathcal{P}$ (Line 8). Then, the probability $\boldsymbol{x}'_k(a_k)$ is set

to zero and $\boldsymbol{x}'_k$ is renormalized to maintain it within the simplex (Lines 9-10). Finally, an alternative action $a_{k'} \neq a_k$ is sampled from $\boldsymbol{x}'_k$ (Line 11). Due to space constraints, further details on Algorithm 1 are in Appendix E.

4 Experiments

To evaluate the performance and scalability of our TSO, we conduct a series of experiments on UNSGs with varying scales and complexities. We compare TSO against two baselines: PSRO and NAL. Methods such as NSGZero/N-FSP (Xue, An, and Yeo 2022; Heinrich and Silver 2016) (extensive-form) and EPSRO (Zhou et al. 2022) (requires explicit action enumeration) are incompatible with our normal-form, non-enumerable setting. Our experiments are performed on a workstation with an Intel i9-14900K CPU and an NVIDIA RTX 4090 GPU with 24GB memory.

4.1 Experimental Setup

Game Environments. We design three sets of game environments based on graph size and action space: small-scale, medium-scale, and large-scale. The detailed configurations are summarized in Appendix G.

Small-Scale Game (S-1). This experiment is conducted on a graph with 16 nodes and 40 edges. It serves as a foundational test case where the action spaces are small enough, allowing us to verify the convergence properties of our algorithm in a controlled setting. The attacker has 92 paths, while the defense team, consisting of two defenders with 11 candidate locations each, has 11^2 joint actions.

Medium-Scale Games (M-1 to M-4). There are four experiments which are based on a graph with 64 nodes and 300 edges and are designed to systematically evaluate the scalability of the algorithms.

- Scenarios **M-1, M-2, and M-3** progressively increase the maximum path length for the attacker from 8 to 10. This expands the attacker's action space from 1955 to 20 029, while the defender's action space remains fixed at 50. This setup specifically tests the algorithms' ability to handle an increase in the attacker's action space.
- Scenario **M-4** builds upon the same graph structure with two defender resources. The maximum path length for the attacker is 7, resulting in 513 possible actions. The defender's action space increases exponentially from 150 to $150^2 = 22{,}500$, creating a game that challenges the defenders' tree-based action representation methods.

Large-Scale Game (L-1). The large-scale experiment is set on a 10000-node graph with 31660 edges. Given a maximum path length of 100 for the attacker in this environment, the number of possible attacking paths is so enormous that the action space cannot be feasibly enumerated. This scenario is designed as a test to evaluate the practical applicability of TSO in situations where action enumeration is infeasible.

Game with Asymmetric Payoffs. In the experimental setup described above, the attacker receives the same reward regardless of which target is reached. To test our algorithm's adaptability to more complex reward structures, we modify the S-1 setup. We configure the game with 4 distinct targets, each offering a different reward to the attacker: [1, 2, 3, 4]. The attacker's maximum path length is 9, resulting in an action space of 120 strategies. The defense team's setup remains the same as in S-1.

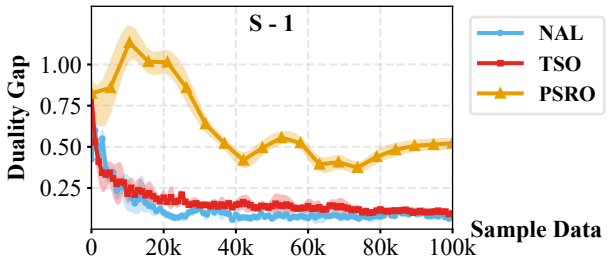

Figure 2: Small-Scale Game Experiment Results.

Game with Decentralized Defenders. We investigate the performance of our algorithm under a decentralized decision-making model for the defense team. The environment is based on the S-1 experiment; however, we remove the assumption of central coordination among defenders. Each defender independently determines its placement decision without knowledge of the actions taken by other defenders, and aims to maximize the overall team utility.

4.2 Baselines and Evaluation

For games of small and medium scale where the action spaces can be enumerated, we use the **duality gap**, as introduced in the preliminaries, as our primary evaluation metric. A lower duality gap indicates a closer approximation to an NE.

We compare TSO with two baselines: NAL and PSRO. Since NAL and TSO use different network architectures, their hyperparameters were tuned separately to ensure optimal performance. Hyperparameter details are in Appendix G.2. For each experiment, we report the mean and standard deviation over multiple runs with different random seeds.

A direct performance comparison between PSRO, NAL and TSO is non-trivial due to their different training paradigms. An "iteration" in PSRO, which involves computing a best response and solving a meta-game, is not equivalent to a "training step" in NAL and TSO. To establish a fair basis for comparison, we align their performance curves using the number of samples collected during training. Specifically, we constrain the total number of samples for PSRO to be identical to that of NAL and TSO. After each PSRO iteration, we evaluate the current policies by calculating their duality gap. This performance metric is then plotted against the cumulative number of samples consumed up to that iteration, enabling a direct alignment with the results of NAL and TSO. For example, as shown in Figure 3, the x-axis of the plot represents the number of samples.

4.3 Experimental Results Analysis

Performance on Small- and Medium-Scale Games. Figure 2 illustrates that in the small-scale **S-1** scenario, TSO achieves convergence performance comparable to NAL, with

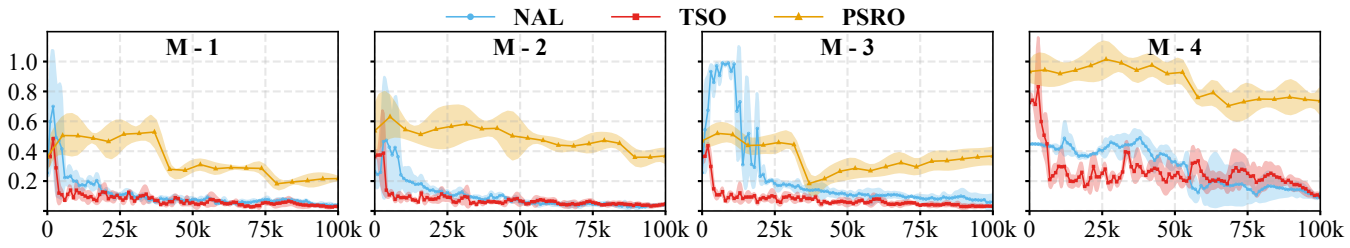

Figure 3: Medium-Scale Games Experiment Results. X-axis: number of sampled data; Y-axis: duality gap.

only a marginal gap. This result is expected, as NAL is known to perform well in environments with smaller action spaces. The slight performance difference is acceptable, likely stemming from the increased optimization complexity introduced by the multiplicative structure in TSO.

As illustrated in Figure 3, across all medium-scale experiments (**M-1** to **M-4**), TSO and NAL exhibit similar duality gaps at convergence. In the case of **M-3**, the TSO method demonstrates a slight advantage over NAL. In all cases, TSO converges faster than NAL. This suggests that as the dimensionality of the action space increases, NAL's efficiency becomes limited. Our tree-based action representation enables more efficient optimization. The PSRO baseline's performance is substantially lower than both TSO and NAL across all S and M-series experiments. This underscores a critical limitation of PSRO in UNSGs: the bias introduced by approximating the best response accumulates over iterations, severely hindering convergence to the NE. We also evaluate the impact of the number of defenders on TSO's performance; details are in Appendix G.4. Further details regarding the large-scale experiment are presented in Appendix G.3.

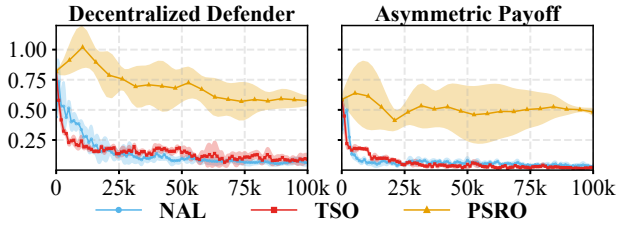

Figure 4: Diverse Games Experiment Results.

Performance in Asymmetric and Decentralized Games. We further evaluate TSO in a game with asymmetric payoffs and another with decentralized defenders, as shown in Figure 4. In these experiments, TSO's performance remains highly competitive with NAL, and both methods maintain a significant advantage over PSRO. This demonstrates the TSO framework's ability to adapt and sustain its strong performance when extending to a wider variety of game structures.

Hyperparameter Sensitivity Analysis and Training Time Comparison. We conduct hyperparameter analysis of TSO with respect to key hyperparameters. More details can be found in Appendix G.5. We also compare the wall-clock time of TSO and PSRO, as shown in Appendix G.6.

4.4 Ablation Studies

Role of Tree-Based Action Representation in TSO. The most significant function of our tree-based action representation is to make stochastic gradient optimization viable in UNSGs with action spaces too large to be feasibly enumerated. These results are demonstrated in the **L-1** experiment and Appendix G.3, where TSO achieves better performance than PSRO in a setting where NAL fails to optimize. The performance advantage of TSO over NAL in Figure 3 also highlights the benefits of our tree-based action representation.

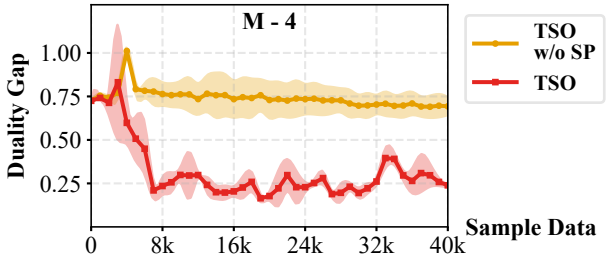

Figure 5: Ablation Experiment Results.

Impact of the Sample-and-Prune Mechanism. To validate the Sample-and-Prune Mechanism, we performed an ablation on the **M-4** setting, comparing TSO to a variant without the mechanism (*TSO w/o SP*). As shown in Figure 5, full TSO converges to a better NE, while the ablated version gets stuck in a local optimum, confirming that the Sample-and-Prune Mechanism is essential for improving TSO's convergence.

5 Conclusion

In this paper, we proposed TSO, a framework that effectively solves large-scale UNSGs. TSO bridges the gap between stochastic optimization for NE-finding and the practical demands of games with massive and combinatorial action spaces. TSO overcomes the challenge of massive action spaces using the tree-based representation for efficient action sampling and a sample-and-prune mechanism to improve solution quality. Extensive experiments confirm that TSO outperforms existing baselines. In summary, TSO provides a scalable and effective solution for a critical class of security games and offers a promising methodology for other large-scale game-theoretic problems.

Acknowledgments

This research is supported by the InnoHK Funding.

References

Duan, Z.; Huang, W.; Zhang, D.; Du, Y.; Wang, J.; Yang, Y.; and Deng, X. 2023. Is Nash equilibrium approximator learnable? In *AAMAS*, 233–241.

Facchinei, F.; and Pang, J.-S. 2003. *Finite-dimensional variational inequalities and complementarity problems*. Springer.

Gemp, I.; Marris, L.; and Piliouras, G. 2024. Approximating Nash equilibria in normal-form games via stochastic optimization. In *ICLR*.

Gemp, I.; Savani, R.; Lanctot, M.; Bachrach, Y.; Anthony, T.; Everett, R.; Tacchetti, A.; Eccles, T.; and Kramár, J. 2022. Sample-based approximation of Nash in large many-player games via gradient descent. In *AAMAS*.

Goktas, D.; Parkes, D. C.; Gemp, I.; Marris, L.; Piliouras, G.; Elie, R.; Lever, G.; and Tacchetti, A. 2022. Generative adversarial equilibrium solvers. In *ICLR*.

Heinrich, J.; and Silver, D. 2016. Deep reinforcement learning from self-play in imperfect-information games.

Iwashita, H.; Ohori, K.; Anai, H.; and Iwasaki, A. 2016. Simplifying urban network security games with cut-based graph contraction. In *AAMAS*, 205–213.

Jain, M.; Conitzer, V.; and Tambe, M. 2013. Security scheduling for real-world networks. In *AAMAS*, 215–222. ISBN 978-1-4503-1993-5.

Jain, M.; Korzhyk, D.; Vaněk, O.; Conitzer, V.; Pěchouček, M.; and Tambe, M. 2011. A double oracle algorithm for zero-sum security games on graphs. In *AAMAS*, 327–334.

Lanctot, M.; Zambaldi, V.; Gruslys, A.; Lazaridou, A.; Tuyls, K.; Pérolat, J.; Silver, D.; and Graepel, T. 2017. A unified game-theoretic approach to multiagent reinforcement learning. In *NeurIPS*, volume 30.

Li, P.; Li, S.; Wang, X.; Cerny, J.; Zhang, Y.; McAleer, S.; Chan, H.; and An, B. 2024. Grasper: A generalist pursuer for pursuit-evasion problems. In *AAMAS*.

Li, S.; Wang, X.; Zhang, Y.; Xue, W.; Černỳ, J.; and An, B. 2023. Solving large-scale pursuit-evasion games using pre-trained strategies. In *AAAI*, volume 37, 11586–11594.

Liu, S.; Marris, L.; Piliouras, G.; Gemp, I.; and Heess, N. 2024. NfgTransformer: Equivariant representation learning for normal-form games. In *ICLR*.

Marris, L.; Gemp, I.; Anthony, T.; Tacchetti, A.; Liu, S.; and Tuyls, K. 2022. Turbocharging solution concepts: Solving NEs, CEs and CCEs with neural equilibrium solvers. In *NeurIPS*, volume 35, 5586–5600.

McMahan, H. B.; Gordon, G. J.; and Blum, A. 2003. Planning in the presence of cost functions controlled by an adversary. In *ICML*, 536–543.

Meng, L.; Chen, W.; Li, W.; Yang, T.; Zhang, Y.; and Gao, Y. 2025. Reducing variance of stochastic optimization for approximating Nash equilibria in normal-form games. In *ICML*.

Nash, J. 1951. Non-cooperative games. *Annals of Mathematics*, 286–295.

Raghunathan, A.; Cherian, A.; and Jha, D. 2019. Game theoretic optimization via gradient-based nikaido-isoda function. In *ICML*, 5291–5300. PMLR.

Tsai, J.; Yin, Z.; Kwak, J.-y.; Kempe, D.; Kiekintveld, C.; and Tambe, M. 2010. Urban security: game-theoretic resource allocation in networked domains. In *AAAI*, volume 24, 881–886.

Xue, W.; An, B.; and Yeo, C. K. 2022. NSGZero: Efficiently learning non-exploitable policy in large-scale network security games with neural monte carlo tree search. In *AAAI*, volume 36, 4646–4653.

Xue, W.; Zhang, Y.; Li, S.; Wang, X.; An, B.; and Yeo, C. K. 2021. Solving large-scale extensive-form network security games via neural fictitious self-play. In *IJCAI*.

Yongacoglu, B.; Arslan, G.; Pavel, L.; and Yuksel, S. 2024. Paths to equilibrium in games. In *NeurIPS*, volume 37, 100870–100894.

Zhang, Y.; An, B.; Tran-Thanh, L.; Wang, Z.; Gan, J.; and Jennings, N. R. 2017. Optimal escape interdiction on transportation networks. In *IJCAI*.

Zhang, Y.; Guo, Q.; An, B.; Tran-Thanh, L.; and Jennings, N. R. 2019. Optimal interdiction of urban criminals with the aid of real-time information. In *AAAI*, volume 33, 1262–1269.

Zhou, M.; Chen, J.; Wen, Y.; Zhang, W.; Yang, Y.; Yu, Y.; and Wang, J. 2022. Efficient policy space response oracles. *arXiv preprint arXiv:2202.00633*.

Zhuang, S.; Li, S.; Yang, T.; Li, M.; Shi, X.; An, B.; and Zhang, Y. 2025. Solving urban network security games: learning platform, benchmark, and challenge for AI Research. *arXiv preprint arXiv:2501.17559*.