# Solving Large-Scale Extensive-Form Network Security Games via Neural Fictitious Self-Play

**Wanqi Xue**[1] , **Youzhi Zhang**[2] , **Shuxin Li**[1] , **Xinrun Wang**[1] , **Bo An**[1] and **Chai Kiat Yeo**[1]
[1]School of Computer Science and Engineering, Nanyang Technological University, Singapore
[2]Department of Computer Science, Dartmouth College, USA
wanqi001@e.ntu.edu.sg, youzhi.zhang@dartmouth.edu, {shuxin.li, xinrun.wang, boan, asckyeo}@ntu.edu.sg

## Abstract

Securing networked infrastructures is important in the real world. The problem of deploying security resources to protect against an attacker in networked domains can be modeled as Network Security Games (NSGs). Unfortunately, existing approaches, including the deep learning-based approaches, are inefficient to solve large-scale extensive-form NSGs. In this paper, we propose a novel learning paradigm, NSG-NFSP, to solve large-scale extensive-form NSGs based on Neural Fictitious Self-Play (NFSP). Our main contributions include: i) reforming the best response (BR) policy network in NFSP to be a mapping from action-state pair to action-value, to make the calculation of BR possible in NSGs; ii) converting the average policy network of an NFSP agent into a metric-based classifier, helping the agent to assign distributions only on legal actions rather than all actions; iii) enabling NFSP with high-level actions, which can benefit training efficiency and stability in NSGs; and iv) leveraging information contained in graphs of NSGs by learning efficient graph node embeddings. Our algorithm significantly outperforms state-of-the-art algorithms in both scalability and solution quality.

## 1 Introduction

How to secure networked infrastructures, e.g., urban city networks, transportation networks, and web networks, has received extensive attention [Jain *et al.*, 2011; Zhang *et al.*, 2017; Zhang *et al.*, 2019; Okamoto *et al.*, 2012]. The problem of deploying a limited number of security resources (controlled by the defender) to protect against an attacker in networked domains can be modeled as Network Security Games (NSGs). We consider a realistic game setting where the players interact sequentially (extensive-form) and the defender makes decisions based on real-time information about the attacker [Zhang *et al.*, 2019]. The objective of NSGs is to find a Nash Equilibrium (NE) policy for the defender. Traditionally, the defender's policy is computed by programming-based NE-solving techniques, e.g., the incremental strategy generation algorithms [Bosansky *et al.*, 2014; Zhang *et al.*, 2019], which start from a restricted game and iteratively expand it until convergence. One important requirement of these approaches is that all of the attacking paths are enumerable, which is to ensure that there is at least a terminal state in the restricted game for each attacking path to make the incremental strategy generation algorithm converge. However, in large-scale NSGs, e.g., real-world road networks, where the number of attacking paths are prohibitively large, programming-based NE-solving approaches tend to lose effectiveness. For example, in the real world, the number of possible attacking paths could be more than $6.6^{18}$ [Jain *et al.*, 2011], which will make it impossible to enumerate all of them due to the limited memory.

Recently, there has been an increasing interest in combining Deep Learning (DL) with game theory for finding NE [Heinrich and Silver, 2016; Lanctot *et al.*, 2017; Brown *et al.*, 2019]. DL-based NE-solving algorithms use Deep Neural Networks (DNNs) to learn states-to-actions mappings for approximating strategies, counterfactual regrets, etc. They usually execute in a sampling style and are able to capture the structure of underlying enormous state spaces by leveraging strong representation ability of DNNs, making them potential for solving large-scale and complex real-life problems.

Unfortunately, existing DL-based NE-solving algorithms are unable to solve large NSGs. In NSGs, players occupy nodes of graphs, e.g., road networks, and can only move to their adjacency nodes (legal actions) at each step. A consequence is that legal actions change with players' current positions or states. To approximate states-to-actions mappings, a naive implementation is to set the output dimension of DNNs equal to the maximum number of legal actions, with each output corresponding to one legal action though the action changes with states. However, this naive setting yields poor results in practice because each output of DNNs has no consistent semantics [Farquhar *et al.*, 2020]. On the other hand, it is infeasible to set the output dimension of DNNs equal to the number of all actions and use masks to filter out all illegal actions at each state. The reason is that, in NSGs, the defender's action space is prohibitively large because it is a combination of all sub-action spaces of the defender's security resources. For example, when there are four security resources deployed on a road network with one hundred nodes, the defender's action set has $100^4$ elements. Obviously, we cannot define the output of DNNs at such a scale.

In this paper, we propose a novel learning paradigm, NSG-NFSP, for approximating an NE policy in large-scale extensive-form NSGs. The method is based on Neural Fictitious Self-Play (NFSP), which intrinsically ensures its convergence. Our main contributions are fourfold. Firstly, we propose to train the best response (BR) policy network in NFSP to be a mapping from action-state pair to action-value, which avoids the aforementioned unachievable requirement where the output of DNN must cover the overall action spaces of NSGs. Secondly, we convert the average policy network into a metric-based classifier, helping an NFSP agent to assign distributions only on legal actions rather than all actions. Thirdly, we propose a framework to enable NFSP with high-level actions, which can enhance training efficiency and stability in NSGs. Finally, we propose to learn efficient graph node embeddings by *node2vec*, to leverage information contained in the graphs of NSGs. We conduct experiments in NSGs played on synthetic networks and real-world road networks. Our algorithm significantly outperforms state-of-the-art algorithms in both scalability and solution quality.

## 2 Preliminaries and Related Works

### 2.1 Network Security Games

Network Security Games (NSGs) are proposed to model the problem of deploying a limited number of security resources to protect against an adaptive attacker in networked domains [Jain *et al.*, 2011]. For example, the police department distributes collaborating police officers to prevent a criminal from escaping or attacking in urban cities [Zhang *et al.*, 2017; Jain *et al.*, 2011]. An NSG is played on a graph $G = (V, E)$ which consists of a set of edges $E$ and a set of nodes $V$. There are two players, the defender and the attacker. The attacker, starting from one of the source nodes $v_0^{att} \in V_s \subset V$, tries to reach one of the target nodes $\chi \in V_t \subset V$ within a fixed time horizon $T$[1]. The defender controls $m$ security resources and dynamically allocates them to catch the attacker before he reaches any of the targets. We assume that the defender can observe the real-time location of the attacker, with the help of advanced tracking technologies such as GPS, but the attacker can only see the initial locations of the security resources [Zhang *et al.*, 2019]. We model NSGs as extensive-form games, where players make decisions sequentially.

At time step $t$, the attacker's state $s_t^{att}$ is a sequence of nodes he has visited, i.e., $s_t^{att} = \langle v_0^{att}, v_1^{att}, \ldots, v_t^{att} \rangle$. $\mathcal{A}_{att} = V$ is the set of attacker actions and $\mathcal{A}_{att}(s_t^{att}) = \{v_{t+1}^{att} | (v_t^{att}, v_{t+1}^{att}) \in E\}$ is the set of legal attacker actions at $s_t^{att}$. For the defender, its state $s_t^{def}$ consists of the attacker's action history (state) and its resources' current locations, i.e., $s_t^{def} = \langle s_t^{att}, l_t^{def} \rangle$ where $l_t^{def} = \langle v_t^0, \ldots, v_t^{m-1} \rangle$. $l_t^{def}$ is adjacent to resource location $\langle v_{t+1}^0, \ldots, v_{t+1}^{m-1} \rangle$ if $(v_t^R, v_{t+1}^R) \in E, \forall R \in \{0, \ldots, m-1\}$. We denote $Adj(l_t^{def})$ as the set of all resource locations which are adjacent to $l_t^{def}$. With this concept, we can define the set of legal defender actions at $s_t^{def}$ as $\mathcal{A}_{def}(s_t^{def}) = Adj(l_t^{def})$ and $\mathcal{A}_{def} = V^m$ is the set of defender actions. For both the attacker and the defender, illegal actions at state $s$ are those actions in $\mathcal{A}$ but not in $\mathcal{A}(s)$. A policy $\pi(s) = \Delta(\mathcal{A}(s))$ describes a player's behavior, where $\Delta(\cdot)$ represents a probability distribution. Both players act simultaneously at each step by sampling actions from their policies: $a \sim \pi(s)$. The attacker is caught if he and at least one of the security resources are in the same node at the same time. A game ends when the attacker either reaches any of the targets within the maximum allowed time or is caught. If the defender successfully protects the targets within the time horzion $T$, she will be awarded with a positive unit utility (an end-game reward) $u_{def} = 1$. Otherwise, no award will be given to the defender. The game is zero-sum, so $u_{att} = -u_{def}$. The worst-case defender utility $\mathcal{E}_{def}(\pi_{def})$ is the expected payoff for the defender (with policy $\pi_{def}$) given that the attacker best responds to it. Formally, $\mathcal{E}_{def}(\pi_{def}) = \min_{\pi_{att}} \mathbb{E}\big[u_{def} | \pi_{def}, \pi_{att}\big]$. $\pi_{def}^*$ is optimal if $\pi_{def}^* \in \arg\max_{\pi_{def}} \mathcal{E}_{def}(\pi_{def})$. The optimality for the attacker is defined similarly. A Nash Equilibrium (NE) is reached if and only if both the defender and the attacker perform the optimal policy. In NSGs, the optimization objective is to learn an NE policy for the defender.

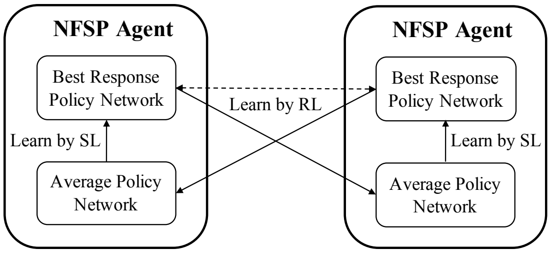


Figure 1: The NFSP framework.

### 2.2 Neural Fictitious Self-Play

**Fictitious play** (FP) [Brown, 1951] is a game-theoretic algorithm for learning NE from self-play. In FP, each agent plays with its opponent's past average policy and best responds against it. **Fictitious Self-Play (FSP)** [Heinrich *et al.*, 2015] extends FP from normal form to extensive form and realizes it in a sampling and machine learning style. **Neural Fictitious Self-Play (NFSP)** [Heinrich and Silver, 2016] combines FSP with neural network function approximation. As in Figure 1, each NFSP agent consists of two neural networks, i.e., the best response (BR) policy network and the average policy network. The BR policy network is trained by reinforcement learning (RL) algorithms, e.g., DQN [Mnih *et al.*, 2015], to maximize the expected total rewards. It considers the opponent as part of the environment. The average policy network is trained to approximate the past average behaviours of the BR policy network by supervised learning (SL). It outputs the probabilities of actions chosen, historically, by the BR policy network. Each NFSP agent behaves according to a mixture of its BR policy and average policy (with a mixing constant $\eta$ which is called anticipatory parameter).

Most applications of NFSP are limited in domains with small discrete action spaces. Despite this, applying NFSP to other types of action spaces has received extensive attention.

[1] The target nodes represent destinations to be attacked or exits to escape.

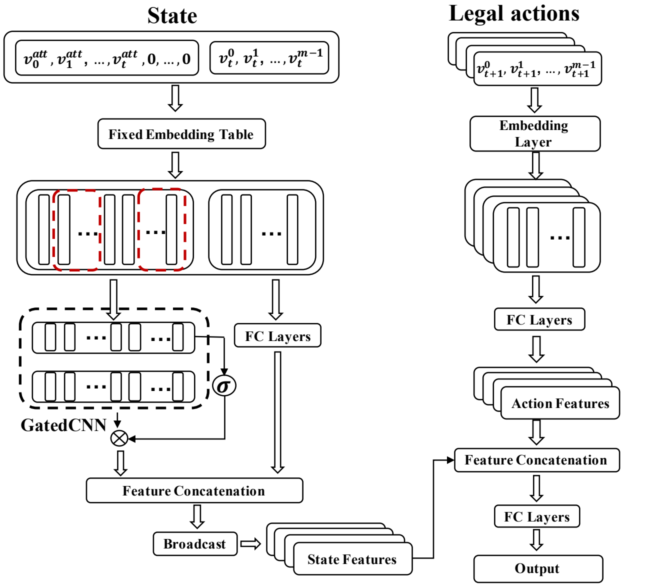


Figure 2: The network structure of the defender (the two red boxes indicate the convolutional blocks in GatedCNN).

OptGradFP [Kamra *et al.*, 2018] firstly introduces fictitious play to continuous action spaces. It applies policy gradient algorithm over policy network which predicts parameters of continuous action distributions. DeepFP [Kamra *et al.*, 2019] generalizes OptGradFP by using flexible implicit density approximators. Currently, applying NFSP to games like NSGs, whose action spaces are large and legal actions vary significantly with states, remains unexplored. The main challenge is that the output of the two DNNs in NFSP cannot cover all actions, and if just covering legal actions, they will lack consistent semantics. With many recent works focusing on adapting deep RL to large discrete action spaces [He *et al.*, 2016; Chandak *et al.*, 2019], we propose our solution for the BR policy network based on DRRN [He *et al.*, 2016]. It models Q-values as an inner product of state-action representation pairs. DRRN is designed for natural languages domain. We adapt it to make it suitable for NSGs whose actions are defined on graph nodes. For the average policy network, our solution is inspired by metric-based few-shot learning [Snell *et al.*, 2017]. We propose to address the problem by transforming actions and states to a space where the probabilities of actions can be determined via comparing some metrics, e.g., cosine similarity. Further discussions about related works are provided in the appendix of the full version.

## 3 Methodology

In this section, we introduce a novel learning paradigm, NSG-NFSP, for solving large-scale NSGs. Note that despite the method being proposed for solving NSGs, the basic ideas can be easily applied to other games, especially those whose legal action spaces vary significantly with states. We provide the overall algorithm in the appendix.

### 3.1 Approximating Best Response Policy

It is essential to properly approximate the BR policy in NFSP because the final (average) policy is supervisedly trained from the behavior of the BR policy network. The BR policy network in the vanilla NFSP algorithm learns a mapping from states to action-values, which internally requires the DNN's outputs to cover all possible actions. However, for games like NSGs, it is impossible to meet this requirement because the overall action space is enormous. To address the problem, we propose to convert the BR policy network to be a mapping from state-action pairs to Q-values. Concretely, we use an action representation network and a state representation network, parameterized by $\theta^Q_\alpha$ and $\theta^Q_\beta$, to extract features from each legal action and state respectively, generating feature vectors $h_a$ and $h_s$. The extracted features $h_a$ and $h_s$ are concatenated and sent to a fully connected network $f(h_a, h_s; \theta^Q_\gamma)$ with parameters $\theta^Q_\gamma$ to predict the action-value. We denote $\theta^Q = \{\theta^Q_\alpha, \theta^Q_\beta, \theta^Q_\gamma\}$ as the parameters of the BR policy network. During training, an agent stores its experienced transition tuples, $(s, a, r, s', \mathcal{A}(s'))$, in a replay buffer $\mathcal{M}_{RL}$, where $r \sim R(\cdot|s, a)$ (reward function) is the immediate reward and $s' \sim P(\cdot|s, a)$ (transition function) is the next state. The BR network parameters $\theta^Q$ are optimized by minimizing the loss:

$$\mathcal{L}(\theta^Q) = \mathbb{E}_{s,a,r,s'}\Big[(r + \max_{a' \in \mathcal{A}(s')} Q(s', a'; \theta^{Q'}) - Q(s, a; \theta^Q))^2\Big] \tag{1}$$

where $\theta^{Q'}$ denotes the parameters of the target network. $\theta^{Q'}$ is periodically copied from $\theta^Q$, while in other cases it is frozen to improve the stability of training.

Figure 2 presents the overall network architecture, NSG-BR, for the defender. We can design the network for the attacker similarly. Since the elements of states and actions are graph nodes, we firstly embed those graph nodes before they can be fed into neural networks. For the action representation network, we use a learnable embedding layer. For the state representation network, we pre-compute the embeddings (the approach is introduced in Section 3.4). After embedding graph nodes, we need to extract features from the attacker's history. Taking into account the speed and effectiveness, we apply a structure similar to GatedCNN [Dauphin *et al.*, 2017] to process these sequential data. Specifically, the sequential data padded to the maximum length is fed into two separate convolutional blocks which have identical structures. The output of one block is activated by the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, and the result serves as the gate to control the output of the other convolutional block. After extracting features for a state, the state feature vector is duplicated (broadcast) $|\mathcal{A}(s)|$ times (number of legal actions at the state) and concatenated with legal action features. Then the state-action pairs' features are passed to several fully connected layers to predict the final state-action values.

### 3.2 Approximating Average Policy

The average policy network in the vanilla NFSP algorithm is a classifier-like network whose output scales linearly with the cardinality of action set $\mathcal{A}$. In NSGs, assigning distribu-

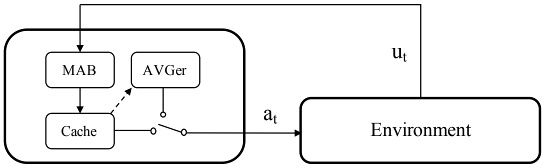


Figure 3: The architecture of NFSP with high-level actions.

tions over legal action set $\mathcal{A}(s)$, rather than the whole action set, is preferred. We propose to convert the average policy network into a metric-based classifier, transforming states and legal actions to a space where they can be compared by some metrics. We use fully-connected layers to learn the metric. The reason is that i) learnable metric is more representative compared to heuristic metric; and ii) we can reuse the network architecture as depicted in Figure 2. Similar to NSG-BR, the average policy network, NSG-AVG, has three parts, i.e., the action representation network, the state representation network and the metric network. We denote $\theta^{\Pi} = \{\theta^{\Pi}_{\alpha}, \theta^{\Pi}_{\beta}, \theta^{\Pi}_{\gamma}\}$ to be parameters of NSG-AVG, where $\theta^{\Pi}_{\alpha}, \theta^{\Pi}_{\beta}, \theta^{\Pi}_{\gamma}$ are the parameters of its three sub-networks respectively. NSG-AVG works similarly to NSG-BR, the main difference being that NSG-AVG assigns probabilities to legal actions rather than Q-values. Let $\Pi(\mathcal{A}(s)|s)$ be the output of NSG-AVG and $\hat{\Pi}(\mathcal{A}(s)|s)$ be the ground truth average BR behaviours. We measure the difference between these two distributions, $\Pi(\mathcal{A}(s)|s)$ and $\hat{\Pi}(\mathcal{A}(s)|s)$, by the expected Kullback-Leibler (KL) divergence. Then the parameters $\theta^{\Pi}$ of NSG-AVG can be optimized by minimizing the difference:

$$\begin{aligned}\mathcal{L}(\theta^{\Pi}) &= -\mathbb{E}_s\Big[\sum_{a\in\mathcal{A}(s)}\hat{\Pi}(a|s)\log\Big(\frac{\Pi(a|s)}{\hat{\Pi}(a|s)}\Big)\Big] \\ &= -\mathbb{E}_{s,a}\Big[\log\Big(\frac{\Pi(a|s)}{\hat{\Pi}(a|s)}\Big)\Big]\end{aligned} \tag{2}$$

The denominator in Eq.(2) can be omitted since it does not depend on $\theta^{\Pi}$. Then the optimization objective becomes minimizing the loss function:

$$\mathcal{L}(\theta^{\Pi}) = -\mathbb{E}_{s,a}\Big[\log\Big(\Pi(a|s)\Big)\Big] \tag{3}$$

An agent records its BR behaviours, i.e., $(s, a)$, in a reservoir buffer $\mathcal{M}_{SL}$ which serves as an expanding dataset. It fits the dataset by applying gradient descent. According to the theoretical convergence of NFSP [Heinrich and Silver, 2016], the average policy network approximates an NE policy.

### 3.3 Enabling NFSP with High-Level Actions

During training, it usually takes many episodes for the BR policy network to approximate the BR policy. For example, in NSGs, the NFSP attacker will do a lot of unnecessary explorations in invalid paths, i.e., the paths which cannot reach any of the targets, before finding the optimal path. This will lead to training inefficiency and instability because i) other agents will play against this weak opponent for a long period; and ii) behaviours of non-BR policy will be used for training the average policy network. The problems can be mitigated if an NFSP agent makes decisions on high-level actions (HLA). For instance, we can force the NFSP attacker in NSGs to make decisions on valid paths or source-target pairs (high-level actions) rather than the next-step location. The idea of HLA is similar to action abstractions [Marino *et al.*, 2019; Lelis, 2020] and options in hierarchical RL [Kulkarni *et al.*, 2016].

To extend the NFSP framework so that an agent can decide on High-Level Actions (NFSP-HLA), we propose to use Multi-Armed Bandit (MAB), a widely used approach to optimize decisions between multiple options (actions), to model the BR policy for an NFSP-HLA agent. Each option of the MAB corresponds to a high-level action. We use a first-in-first-out (FIFO) buffer with length $k$ to record the most recent $k$ game results (utilities). The estimated action value for a high-level action $\zeta$ after $n$ episodes becomes:

$$\hat{Q}_n(\zeta) = \frac{\sum_{j=\tau}^{n} u_j \mathbb{I}[\zeta_j = \zeta]}{\sum_{j=\tau}^{n} \mathbb{I}[\zeta_j = \zeta]} \tag{4}$$

where $\tau = \max(n-k, 1)$, $u_j$ is the player's utility for the $j$-th episode, and $\mathbb{I}$ is binary indicator function. We design two auxiliary modules to fit the MAB best responsor into the framework of NFSP: i) the Averager (AVGer) module, which is used to measure the average policy, by counting the frequency of each high-level action; and ii) the Cache module, which is to temporarily store behaviours of the MAB. Data stored in the Cache is used to update the AVGer.

**Learning Process.** As in Figure 3, before each episode, an NFSP-HLA agent samples its behaviour pattern, acting as either the MAB (with probability $\eta$) or the AVGer (with probability $1-\eta$). If acting as the MAB, the agent chooses the high-level action with the largest estimated value and stores the high-level action in the Cache. Otherwise, the agent samples an high-level action according to the distribution in the AVGer. After confirming the high-level action, the agent interacts with the environment (containing the opponents) and receives an utility at the end of a game. Then the utility is used for training the MAB in accordance with Eq. (4). The Cache pours its records into the AVGer in a fixed frequency, after which it clears itself. By using the Cache, we can avoid rapid changes in the AVGer, thus reducing instability.

**Additional Exploration.** An NFSP-HLA agent does exploration by acting as the AVGer (not the MAB). Such mechanism may lead to suboptimality when some actions are dominated. For example, if the AVGer explores some good actions, the MAB is likely to choose them because of their high estimated values. Behaviours of the MAB will be recorded by the AVGer, further increasing these actions' occurrence frequency (selected by the AVGer). This may result in some actions appearing rarely, and the MAB cannot precisely estimate those actions' values. To overcome this, we design additional exploration for NFSP-HLA agent: if the agent does not act as the MAB, it will perform additional sampling to decide whether to explore or not. If the sampling result indicates exploration, the agent will act randomly and the transitions for this episode will not be recorded by the opponent; Otherwise, the agents will interact normally. Additional exploration confirms that each high-level action occurs enough times that the MAB can conduct action-value estimation.

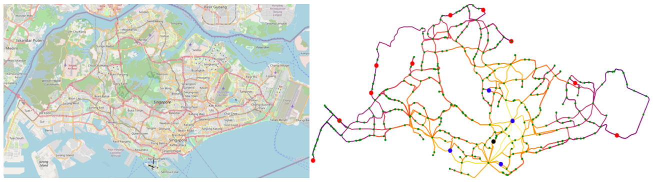


Figure 4: Singapore map and the extracted road network.

### 3.4 Efficient Graph Node Embeddings

To leverage information, e.g., adjacency and connectivity, contained in graphs of NSGs, we propose to use *node2vec* [Grover and Leskovec, 2016], a semi-supervised learning algorithm, to learn representations for nodes. Concretely, we learn a mapping function to transform nodes to low-dimensional space of features which maximizes the likelihood of preserving graph structures. Let $g : V \to \mathbb{R}^D$ be the mapping function, we need to optimize:

$$\max_{g} \sum_{v \in V} \log Prob(N_O(v)|g(v)) \tag{5}$$

where $N_O(\cdot)$ is a randomized procedure that samples many different neighborhoods of a given node. Concretely, the procedure $N_O(\cdot)$ is parameterized by a sampling strategy $O$:

$$O(v_t = x|v_{t-1} = u) = \begin{cases} \frac{U(u,x)}{Z} & \text{if } (u,x) \in E \\ 0 & \text{else} \end{cases} \tag{6}$$

where $U(\cdot,\cdot)$ is the unnormalized transition probability, and $Z$ is the normalizing constant. The sampling strategy $O$ could be, for example, Breadth-First Sampling (BFS), Depth-First Sampling (DFS), etc., generating different transition probabilities. We follow *node2vec* to use a biased $2^{nd}$ order random walk as the sampling strategy. In our unweighted road network graph,

$$U(u,x) = \alpha_{pq}(u,x) = \begin{cases} \frac{1}{p} & \text{if } d_{wx} = 0 \\ 1 & \text{if } d_{wx} = 1 \\ \frac{1}{q} & \text{if } d_{wx} = 2 \end{cases} \tag{7}$$

where $p$, $q$ are two parameters of the random walk which control how fast the walk explores, $w$ is the predecessor of $u$, and $d_{wx}$ is the distance between nodes $w$ and $x$.

After obtaining the mapping function $g$, we use it to embed nodes when extracting features from a state. For action representation, we use a learnable embedding layer. We apply this setting because it can keep flexibility in learning while leveraging graph information.

### 3.5 Discussion about Other Applicable Games

The proposed method is suitable for solving games whose legal actions vary significantly with states. Since the legal actions at different states can be very different, the overall action spaces of these games are usually enormous, leading to inefficiency of existing approaches. Our method is a mitigation for the problem. Apart from NSGs, our method can be applied to games that involve high-dimensional control. In such games, the overall legal action space is a combination of each dimension's legal action space, changes of legal actions in each dimension will accumulate and may lead to a significant change in the overall legal action space. Team-Goofspiel (Section 4.3) is an example for that. Another potential application scenario of our method is text-based games whose actions are defined based on natural languages, i.e., the action spaces are composed of sequences of words from a fixed size and large dictionary. There are constraints on actions at each state to generate a meaningful command (sequence of words), e.g., “open door”, “turn left”. For different states, e.g., ”open” or ”turn”, the corresponding legal action spaces are very different (“door, box, book, . . . ” or “left, right, . . . ”). Our method can be applied to them by firstly learning states and legal action representations and then mapping state-action pair representations to values which represent Q-values or probabilities. High-level actions can be defined, for example, as phrases or sentences, and learning embeddings for each word is also reasonable.

## 4 Experimental Evaluation

We firstly evaluate our algorithm on large-scale NSGs. Then, we perform ablation studies to understand how each component of NSG-NFSP affects the results. Finally, we justify the adaptability of our method to games in other domains. Experiments are performed on a server with a 10-core 3.3GHz Intel i9-9820X CPU and an NVIDIA RTX 2080 Ti GPU.

### 4.1 Large-Scale NSGs

We evaluate our algorithm in NSGs played on both artificially generated networks and real-world road networks.

**Artificially Generated Networks.** We generate the evaluation network by the grid model with random edges [Peng *et al.*, 2013]. Concretely, we sample a $15 \times 15$ grid whose horizontal/vertical edges appear with probability 0.4 and diagonal edges appear with probability 0.1. We set the initial location of the attacker at the center of the grid, and let the defender controls 4 security resources which are distributed uniformly on the network at the beginning[2]. There are 10 target nodes located randomly at the border. We set the time horizon $T$ as 70, 90, and 300, to create NSGs with different scales.

We compare our algorithm with two heuristic defender policies, namely uniform policy and greedy policy, as well as a state-of-the-art algorithm, IGRS++ [Zhang *et al.*, 2019]. For the uniform policy, the defender assigns equal probability to each legal action at a state. For the greedy policy, all of the defender's security resources always move along the shortest path to the attacker's current location. When evaluating the performance, because the game sizes are very large, it is intractable to calculate exact worst-case defender utility. We overcome this by using approximate worst-case defender utility. Specifically, we use a DQN attacker to best respond to the defender and calculate defender utility under this scenario to approximate the worst-case defender utility. We train the DQN attacker for $2 \times 10^5$ episodes, store the best model, and load the best model to play with the defender

[2] NSG-NFSP allows the players to do stochastic initialization, by taking the initialization as the first-step action.

| | NSG-NFSP | Uniform policy | Greedy Policy | IGRS++ |
|---|---|---|---|---|
| T=70 | **0.1770 ± 0.0168** | 0.0730 ± 0.0114 | 0 ± 0 | OOM |
| T=90 | **0.1205 ± 0.0143** | 0.0715 ± 0.0111 | 0 ± 0 | OOM |
| T=300 | **0.0825 ± 0.0120** | 0.0675 ± 0.0110 | 0 ± 0 | OOM |

(a) Synthetic network

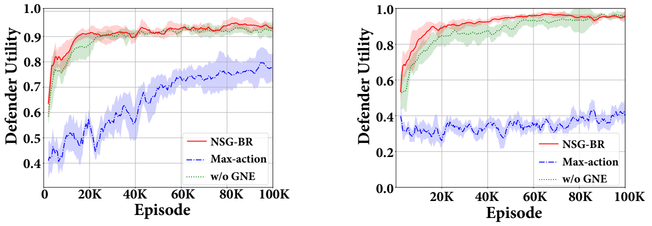
| | NSG-NFSP | Uniform Policy | Greedy Policy | IGRS++ |
|---|---|---|---|---|
| T=30 | **0.1225 ± 0.0140** | 0.0230 ± 0.0066 | 0 ± 0 | OOM |
| T=300 | **0.0720 ± 0.0113** | 0.0055 ± 0.0032 | 0 ± 0 | OOM |

(b) Real-world road network

Table 1: Approximate worst-case defender utilities in NSGs with different time horizons. The "±" indicates 95% confidence intervals over the 2000 testing episodes. OOM stands for Out of Memory.

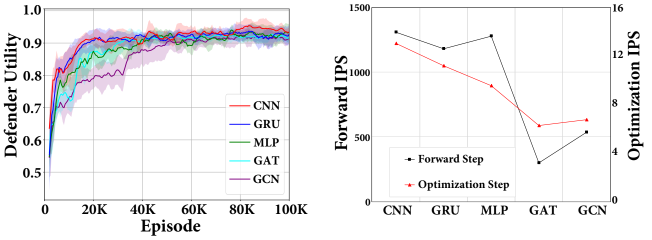
for another 2000 episodes to obtain the final results. We run the NSG-NFSP for $5 \times 10^6$ episodes, which takes around 3-4 days. Despite that the method consumes non-trivial resources when training, it can realize real-time inference, making its deployment possible. Neural network structures and hyperparameters are included in the appendix. As in Table 1a, our method, NSG-NFSP, outperforms the baselines in all three large-scale NSGs. The state-of-the-art algorithm, IGRS++, cannot execute. The reason is that, as an incremental strategy generation algorithm, IGRS++ requires the attacking paths to be enumerable. However, for all of the three settings, it is impossible to enumerate attacking paths. We try to enumerate attacking paths for the case $T = 70$ on a machine with 32G RAM, but after all the memory is occupied, the enumeration does not end. Applying counterfactual regret minimization (CFR) [Zinkevich *et al.*, 2008] or its variants [Schmid *et al.*, 2019; Brown *et al.*, 2019] to solve large-scale NSGs is also infeasible, and we discuss about it in the appendix. Note that the approximate worst-case defender utilities for the greedy policy are always 0, which means that the DQN attacker can always find at least one path to a target node such that the defender with greedy policy cannot prevent him.

**Real-World Road Networks.** As in Figure 4, we extract highways, primary roads and the corresponding intersections from Singapore map via OSMnx [Boeing, 2017]. There are 372 nodes and 1470 edges. Edges are colored according to closeness centrality. The brighter the color, the closer the edges are to the center. The initial position of the attacker, marked in the dark point, locates near the center. There are 4 security resources, marked in blue points. Those target nodes, denoting exits of the map, are marked in red points. We test out that $T = 30$ will lead to attacking/escaping paths unenumerable, and we set the time horizon at 30 and 300. The NFSP defender is trained for $5 \times 10^6$ episodes which takes around a week to finish. We keep the evaluation settings the same as in synthetic networks for $T = 30$ because we find the settings work well. For $T = 300$, where training a DQN attacker is more difficult, we fine-tune hyperparameters to enhance the DQN attacker's performance. As presented in Table 1b, our method significantly outperforms the baselines in both settings. Additional experiments on the Manhattan map also show that our method outperforms the baselines. More experiment results are in the appendix.



(a) 7 × 7 grid (T=7) (b) 15 × 15 grid (T=15)

Figure 5: The learning curves of the defender against an attacker with uniform policy on synthetic networks, averaged across 5 runs.



(a) The learning curves (b) Iterations per second

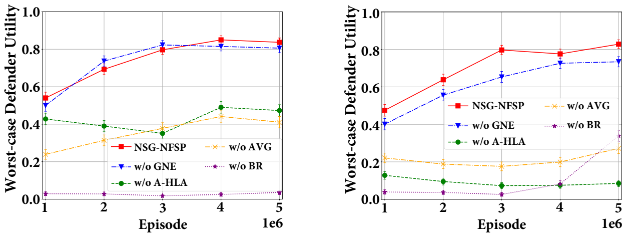
Figure 6: Performance of sequence encoders on the 7 × 7 grid.

## 4.2 Ablation Studies

We perform ablation studies on 7 × 7 and 15 × 15 randomly generated grids, with $T$ at 7 and 15, respectively.

**Best Response Approximation.** We try to evaluate whether the proposed network architecture, NSG-BR, is able to enhance best response approximation. We set the policy of the attacker to be uniform, and let the defender best respond to him. We compare NSG-BR with a naive implementation, max-action DQN, which fixes the output dimension of BR policy network equal to the maximum number of legal actions. As in Figure 5, the performance of max-action DQN is significantly worse than NSG-BR in both settings. We further explore the effect of pre-defined graph node embedding (GNE). We replace GNE with a learnable embedding layer (w/o GNE). Results show that, in simple graph (the 7 × 7 grid), the learning curves have no obvious difference. However, in complex graph (the 15 × 15 grid), GNE does benefit the training. If created properly, GNE can not only speed up the training but also make the process more stable.

**Comparisons of Different Sequence Encoders.** We compare the performance of different types of sequence encoders for extracting state features. The encoders include: i) GatedCNN (CNN), a CNN-based approach (what we use in NSG-NFSP); ii) Gated Recurrent Unit (GRU) [Chung *et al.*, 2014]; iii) Graph Convolution Network (GCN) [Kipf and Welling, 2017]; iv) Graph Attention Network (GAT) [Veličković *et al.*, 2018]; and v) Multi-Layer Perceptron (MLP). We evaluate the average defender utility on the synthetic 7 × 7 grid, setting the policy of the attacker to be uniform. The learning curves of the defender are presented in Figure 6a. We can find that GatedCNN obtains superior performance over other



(a) 7 × 7 grid (T=7)

(b) 15 × 15 grid (T=15)

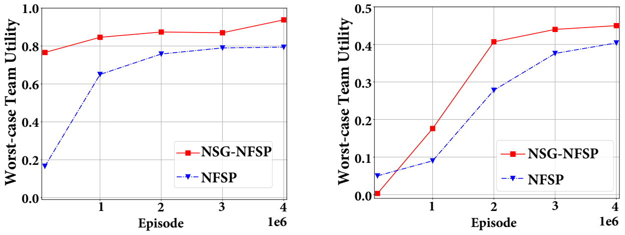
Figure 7: Ablation studies regarding each component of NSG-NFSP.

sequence encoders. We further test the running speed by measuring number of iterations per second (IPS) for both the forward process and the optimization process (batchsize=256). The forward process affects the speed to collect experiences, and the optimization process affects the speed to optimize parameters of DNNs. As in Figure 6b, GatedCNN is the fastest of all the sequence encoders.

**Worst-Case Defender Utility.** To show how each component of NSG-NFSP affects the performance, we replace i) best response policy approximation module (Section 3.1) with max-action DQN (w/o BR); ii) average policy approximation module (Section 3.2) with max-action network (w/o AVG); iii) the attacker in high-level control (Section 3.3) with the original low-level implementation (w/o A-HLA); and iv) graph node embeddings (Section 3.4) with a learnable embedding layer (w/o GNE). As in Figure 7, replacing any of BR, AVG and A-HLA will cause a dramatic drop in performance. As for GNE, it shows a slight performance improvement in simple networks (the 7×7 grid), but in complex networks (the 15 × 15 grids), the enhancement is significant. NSG-NFSP achieves a performance of around 0.8 for the both games. Since the worst-case defender utility is upper-bounded by 1 (the defender wins the game definitely), the results demonstrate near-optimal solution quality of our method.

### 4.3 Adaptability

Our method is suitable for games whose legal action spaces or their sizes vary significantly with states, including but not limited to NSGs. To justify this, we conduct experiments on an extended version of Goofspiel [Ross, 1971], a popular poker game. Specifically, we modify Goofspiel to be played between a team with several members and a single player, as opposed to two single players. We name the modified game as Team-Goofspiel. In $k$-rank $n$-member Team-Goofspiel, there are $k$ rounds and the team consists of $n$ members. At each round $t$ ($t = 1, \ldots, k$), $n$ team members and the single player place bids for a prize of value $t$. The possible bids are $1, \ldots, k$ and each bid can be placed exactly once (the action space of the team player decays rapidly with respect to $t$, containing $(k + 1 - t)^n$ legal actions in total). The player with higher bid wins the prize of the current round; if the bids are equal, no player wins the prize. Both players can only observe the outcome of each round but not the bids. The single player will win the game if its total prize is greater than the team's prize at the end of the game, otherwise the team wins. The winner will obtain a utility of 1. We design two game modes



(a) MAX mode

(b) AVERAGE mode

Figure 8: The worst-case team utility in 4-rank 2-member Team-Goofspiel under MAX and AVERAGE game modes.

for Team-Goofspiel, namely the MAX mode and the AVERAGE mode, where the bid of the team is determined by taking the maximum or the average of all team members' bids, respectively. We set $k = 4$ and $n = 2$ in the experiments. As in Figure 8, our method obtains superior performance compared to the vanilla NFSP algorithm in both the MAX mode and the AVERAGE mode, demonstrating its good adaptability[3].

## 5 Conclusions

In this paper, we propose a novel learning paradigm, NSG-NFSP, for finding NE in large-scale extensive-form NSGs. The algorithm trains DNNs to map state-action pairs to values, which may represent Q-values or probabilities. It enhances the performance by enabling the NFSP attacker with high-level actions and learning efficient graph node embeddings. Our method significantly outperforms state-of-the-art algorithms in both scalability and solution quality.

## Acknowledgements

This research is partially supported by Singtel Cognitive and Artificial Intelligence Lab for Enterprises (SCALE@NTU), which is a collaboration between Singapore Telecommunications Limited (Singtel) and Nanyang Technological University (NTU) that is funded by the Singapore Government through the Industry Alignment Fund – Industry Collaboration Projects Grant.

## References

[Boeing, 2017] Geoff Boeing. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139, 2017.

[Bosansky *et al.*, 2014] Branislav Bosansky, Christopher Kiekintveld, Viliam Lisy, and Michal Pechoucek. An exact double-oracle algorithm for zero-sum extensive-form games with imperfect information. *Journal of Artificial Intelligence Research*, 51:829–866, 2014.

[3] Despite that it is easy to define high-level actions, e.g., bids sequences, and learning bids embeddings to reflect their numerical relationships is reasonable, we omit these in our implementation because they are domain-specific.

[Brown *et al.*, 2019] Noam Brown, Adam Lerer, Sam Gross, and Tuomas Sandholm. Deep counterfactual regret minimization. In *ICML*, pages 793–802, 2019.

[Brown, 1951] George W Brown. Iterative solution of games by fictitious play. *Activity Analysis of Production and Allocation*, 13(1):374–376, 1951.

[Chandak *et al.*, 2019] Yash Chandak, Georgios Theocharous, James Kostas, Scott Jordan, and Philip S Thomas. Learning action representations for reinforcement learning. In *ICML*, pages 941–950, 2019.

[Chung *et al.*, 2014] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *NeurIPS*, 2014.

[Dauphin *et al.*, 2017] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *ICML*, pages 933–941, 2017.

[Farquhar *et al.*, 2020] Gregory Farquhar, Laura Gustafson, Zeming Lin, Shimon Whiteson, Nicolas Usunier, and Gabriel Synnaeve. Growing action spaces. In *ICML*, pages 4335–4346, 2020.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *SIGKDD*, pages 855–864, 2016.

[He *et al.*, 2016] Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. Deep reinforcement learning with a natural language action space. In *ACL*, pages 1621–1630, 2016.

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

[Heinrich *et al.*, 2015] Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *ICML*, pages 805–813, 2015.

[Jain *et al.*, 2011] Manish Jain, Dmytro Korzhyk, Ondřej Vaněk, Vincent Conitzer, Michal Pěchouček, and Milind Tambe. A double oracle algorithm for zero-sum security games on graphs. In *AAMAS*, pages 327–334, 2011.

[Kamra *et al.*, 2018] Nitin Kamra, Umang Gupta, Fei Fang, Yan Liu, and Milind Tambe. Policy learning for continuous space security games using neural networks. In *AAAI*, pages 1103–1112, 2018.

[Kamra *et al.*, 2019] Nitin Kamra, Umang Gupta, Kai Wang, Fei Fang, Yan Liu, and Milind Tambe. Deep fictitious play for games with continuous action spaces. In *AAMAS*, pages 2042–2044, 2019.

[Kipf and Welling, 2017] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017.

[Kulkarni *et al.*, 2016] Tejas D Kulkarni, Karthik R Narasimhan, Ardavan Saeedi, and Joshua B Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *NeurIPS*, pages 3675–3683, 2016.

[Lanctot *et al.*, 2017] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *NeurIPS*, pages 4190–4203, 2017.

[Lelis, 2020] Levi H. S. Lelis. Planning algorithms for zero-sum games with exponential action spaces: A unifying perspective. In *IJCAI*, pages 4892–4898, 2020.

[Marino *et al.*, 2019] Julian RH Marino, Rubens O Moraes, Claudio Toledo, and Levi HS Lelis. Evolving action abstractions for real-time planning in extensive-form games. In *AAAI*, pages 2330–2337, 2019.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[Okamoto *et al.*, 2012] Steven Okamoto, Noam Hazon, and Katia P Sycara. Solving non-zero sum multiagent network flow security games with attack costs. In *AAMAS*, pages 879–888, 2012.

[Peng *et al.*, 2013] Wei Peng, Guohua Dong, Kun Yang, and Jinshu Su. A random road network model and its effects on topological characteristics of mobile delay-tolerant networks. *IEEE Transactions on Mobile Computing*, 13(12):2706–2718, 2013.

[Ross, 1971] Sheldon M Ross. Goofspiel—the game of pure strategy. *Journal of Applied Probability*, pages 621–625, 1971.

[Schmid *et al.*, 2019] Martin Schmid, Neil Burch, Marc Lanctot, Matej Moravcik, Rudolf Kadlec, and Michael Bowling. Variance reduction in monte carlo counterfactual regret minimization (vr-mccfr) for extensive form games using baselines. In *AAAI*, pages 2157–2164, 2019.

[Snell *et al.*, 2017] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *NeurIPS*, pages 4077–4087, 2017.

[Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[Zhang *et al.*, 2017] Youzhi Zhang, Bo An, Long Tran-Thanh, Zhen Wang, Jiarui Gan, and Nicholas R Jennings. Optimal escape interdiction on transportation networks. In *IJCAI*, 2017.

[Zhang *et al.*, 2019] Youzhi Zhang, Qingyu Guo, Bo An, Long Tran-Thanh, and Nicholas R Jennings. Optimal interdiction of urban criminals with the aid of real-time information. In *AAAI*, pages 1262–1269, 2019.

[Zinkevich *et al.*, 2008] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *NeurIPS*, pages 1729–1736, 2008.