Algorithms for Subgame Abstraction with Applications to Cyber Defense

Anjon Basak[1] ✉[0000−0001−8427−038X], Marcus Gutierrez[1][0000−0003−0746−5137], and Christopher Kiekintveld[1][0000−0003−0615−9584]

The University of Texas at El Paso, 500 W University Ave, El Paso, TX 79968, USA
abasak@miners.utep.edu, mgutierrez22@miners.utep.edu,
cdkiekintveld@utep.edu

Abstract. It is typically infeasible to use automated intrusion detection systems to scan every single host in a network with high sensitivity and frequency due to high costs and large network sizes. We present a game-theoretic model between a network administrator and a worm using normal form games with a particular structure where the network admin wants to maximize the security of the network using limited resources, and the attacker wants to infect the network without getting caught. However, a large number of hosts in a network can result in a massive game, making it problematic to compute standard solutions like Nash equilibrium. We propose an abstraction approach for solving large games that have a subgame structure and show that it can be used to solve much larger instances of this cybersecurity scenario than standard algorithms.

Keywords: Cybersecurity, Game theory, abstraction, Nash Equilibrium, solution quality

1 Introduction

Most real-world networks are divided into subnets to increase performance and security, but there are limited resources to inspect/harden devices against attacks. Automated Intrusion Detection Systems (IDS) [17] [13] are an essential defense, but it may not be possible to use a costly IDS on every network host [1]. In a network, botnets often spread easily within a subnet using worms that exploit open ports and unpatched vulnerabilities. However, spreading between subnets requires moving through more secure and highly monitored routers that limit connectivity. This locality leads a game model with a particular structure in a Normal Form Game (NFG). We present a game-theoretic model based on this cyberdefense scenario using an NFG for stopping the spread of an attacker (e.g., a botnet) through a network that has a subnet architecture. In this game model, the network administrator acts as the defender and a worm acts as the attacker. The network administrator wants to use his defense mechanism to stop the spread of a botnet by hardening the security in one or more hosts.

While NFG is a very general representation, it is often problematic to solve an NFG for real-world scenarios because enumerating all possible strategies results in an extremely large game. For example, in an enterprise network the large number of hosts and interconnections can lead to intractable NFG models. Solving an NFG is known to

be a computationally hard problem [6], and most existing algorithms (e.g., implemented in Gambit [14]) do not scale well in practice. An increasingly common approach is to apply some form of automated abstraction to simplify the game. The simplified game is then analyzed using an available solver, and the solution is mapped back into the original game. If the reduced game can retain the vital strategic features of the original game, then in principle the solution of the simpler game may be a reasonable approximation of the solution to the original game.

This general approach has been successful in developing computer poker agents (e.g., [8–11, 18]). Brown et al. used CFR [18] and imperfect recall abstraction with earth mover's distance [7] for a hierarchical abstraction [4] technique. Another widespread approach to handle massive games is Double Oracle (DO) Algorithm [3] [15] which relies on the concept of column/constraint generation techniques.

Two works very closely related to an NFG reduction are one by Conitzer et al. [5] and another one by Bard et al. [2]. In the former paper, the authors gave an abstraction technique which can be used in a class of NFGs called *Any Lower Action Gives Identical Utility* (ALAGIU). The authors show that their technique can be applied recursively in ALAGIU games to abstract the game and find approximate Nash equilibrium. Motivated by the approach, we introduce an abstraction technique we call Iterative Subgame Abstraction and Solution Concept (ISASC). To evaluate this technique, we use a class of NFGs where some actions give identical utility that we call *Approximately Identical Outside Subgames* (AIOS). We also introduce a Pure Strategy Nash Equilibrium solution concept called *Minimum Epsilon Bound* (MEB).

Our main contributions are as follows: (1) we model a cyberdefense scenario using NFG that naturally leads to games with AIOS structure, (2) we offer sophisticated non-iterative and iterative algorithms for solving games using abstraction with both exact and noisy AIOS structure, (3) we present experimental evaluation of our algorithms on both generic games and games based on a cyberdefense scenario, showing that our algorithms substantially improve scalability over baseline equilibrium solution algorithms.

2 Games with AIOS Structure

A *Normal Form Game* (NFG) is a standard representation in game theory in which the outcomes of all possible combinations of strategies are represented using a payoff matrix. The tuple (N, A, u) represents a finite N-player NFG [16], where each player is indexed by i. The set of actions (pure strategies) is given by $A = A_1 \times ... \times A_N$, where A_i is the set of actions for player i. Each vector $a = (a_1, ..., a_N) \in A$ is an action profile. An action (kth) for player i is represented by $a_{i,k}$. We extend to mixed strategies $s_i \in S_i$, and use the notation $\pi^i(a_i)$ to refer to the probability of playing action a_i for player i. Each player has a real-valued utility (payoff) function $u = (u_1, ..., u_N)$ where $u_i : A^N \rightarrow \mathbb{R}$, extended to mixed strategies as usual by using expected utility.

We will consider abstracted games represented as (simpler) NFGs. For these games, we use the same notation but with a hat to denote that it is an abstracted game, $(\hat{N}, \hat{A}, \hat{u})$. We also use $A_i(O)$ to refer to the set of available actions for player i in NFG O. The set of clusters for player i is denoted using $c_i = \{c_{i,1}, ..., c_{i,m}\}$, where $c_{i,m} \subset A_i$ and

$c_{i,m}$ is the m^{th} cluster for player *i*, and $c_{i,m} = \{a_{i,1}, ..., a_{i,k}\}$. Every action belongs to exactly one cluster, so $c_{i,1} \cap c_{i,2} \cap ... \cap c_{i,k} = \emptyset$.

We now introduce a game structure based on the idea of forming subgames with strong interactions within a subgame, but weak interactions outside of the subgame. We call this *Approximately Identical Outside Subgames* (AIOS), as shown in Figure 1. The fundamental idea is to create clusters of strategies for both players that form subgames. Within a subgame, the strategies and payoffs can vary arbitrarily. However, outside of the subgame, the strategies for each player should have payoffs as similar as possible for playing against any opponent strategy, not in the subgame. Games with exact AIOS have identical payoffs outside the subgame, while games with noisy AIOS weaken this to allow some variation in the payoffs outside the subgames.

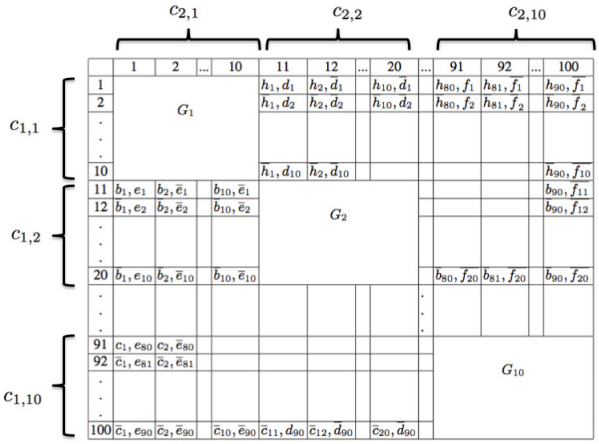


Fig. 1: AIOS Structure in an NFG

In Figure 1, suppose the row player is player 1 and column player is player 2. If player 1 decides to play any strategy from $\{1-10\} \in c_{1,1}$, he needs to worry only about the probabilities assigned by player 2 to strategies $\{1-10\} \in c_{2,1}$. Intuitively this is because if player 2 plays from strategies outside of $c_{2,1}$ the payoff is the same for the row player no matter which action he chooses among $\{1-10\} \in c_{1,1}$. The subgame G_1 is formed by considering only actions in $c_{1,1}$ and $c_{2,1}$.

3 A Cyber Defense Game with AIOS

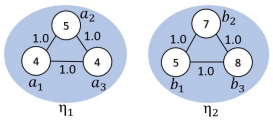


Fig. 2: Example network with $t_{i,j} = 1$ and $T(\eta_k, \eta_l) = 0$

We now present a cybersecurity scenario for a *botnet* attack where the AIOS (Section 2) structure arises naturally. Figure 2 shows an example with 2 subnets containing 3 nodes each. A network is a collection of nodes that belong to exactly 1 subnet η_k. Every host has a value v_i. $t_{i,j}$ represents the *intra-transmission probability* for the botnet to propagate from node i to j within the same subnetwork η_k. The *inter-transmission*

probabilities, represented by $T(\eta_k, \eta_l)$, is for the botnet propagating from subnet η_k to η_l. The botnet spreads on a new subnet η_l from current subnet η_k and infects the nodes of subnet η_l like a worm maintaining the intra-transmission probabilities. We model a one-shot game where the Defender selects a node i to *defend* (e.g., closing ports, patching vulnerabilities, increasing monitoring). The defend action reduces the transmission probabilities for all edges connected to i and stops any attack that spreads to node i. The attacker selects an initial node to attack, which spreads according to the transmission probabilities (which can be estimated using simulation).

If the botnet spreads to a defended node and is detected, the Defender pays a cost equal to the total value of the infected nodes (to clean up the attack), but the attacker receives a payoff of zero. If the attack does not interact with a defended node, the attacker receives the sum of the values of all the infected nodes. We estimate the payoff matrix for a particular game using Monte Carlo simulation to estimate the spread of the infection for each pair of strategies.

Attacker

Defender	a_1	a_2	a_3	b_1	b_2	b_3
a_1	-4.0, 0.0	-7.01, 2.27	-6.47, 2.24	-20.0, 20.0	-20.0, 20.0	-20.0, 20.0
a_2	-5.97, 1.97	-5.0, 0.0	-8.0, 2.02	-20.0, 20.0	-20.0, 20.0	-20.0, 20.0
a_3	-9.0, 2.19	-9.0, 2.29	-4.0, 0.0	-20.0, 20.0	-20.0, 20.0	-20.0, 20.0
b_1	-13.0, 13.0	-13.0, 13.0	-13.0, 13.0	-5.0, 0.0	-10.99, 3.75	-11.46, 3.69
b_2	-13.0, 13.0	-13.0, 13.0	-13.0, 13.0	-8.97, 3.18	-7.0, 0.0	-13.0, 3.2
b_3	-13.0, 13.0	-13.0, 13.0	-13.0, 13.0	-12.0, 2.99	-12.0, 3.08	-8.0, 0.0

(a)

Attacker

Defender	a_1	a_2	a_3	b_1	b_2	b_3
a_1	-4.0, 0.0	-7.62, 3.09	-7.12, 3.04	-20.7, 18.58	-20.69, 18.53	-20.7, 18.44
a_2	-6.83, 2.97	-5.0, 0.0	-8.82, 2.93	-20.88, 18.48	-20.84, 18.45	-20.89, 18.44
a_3	-8.6, 2.38	-8.66, 2.33	-4.0, 0.0	-19.89, 18.66	-19.93, 18.77	-19.91, 18.79
b_1	-12.97, 12.17	-12.96, 12.15	-12.96, 12.18	-5.0, 0.0	-10.64, 3.96	-11.28, 4.01
b_2	-14.01, 12.36	-14.02, 12.34	-14.03, 12.38	-9.39, 4.09	-7.0, 0.0	-13.14, 4.06
b_3	-14.41, 12.33	-14.45, 12.26	-14.44, 12.3	-12.39, 3.86	-12.41, 4.08	-8.0, 0.0

(b)

Fig. 3: (a) Game for Figure 2 with $t_{i,j} = 1$ and $T(\eta_k, \eta_l) = 0$. (b) Game for Figure 2 with $t_{i,j} = [0.85, 1.0]$ and $T(\eta_k, \eta_l) = 0.10$

Figure 3(a) shows the NFG representation for the example in Figure 2 assuming $t_{i,j} = 1$ and no edges exist between subnets. In this case, the game has an exact AIOS structure. When the two players play on the same subnet, there is a strategically interesting game. However, when the two players play outside of the same subnets, there is no interaction. Intuitively, this is because the Defender will never be able to detect the Attacker's botnet because no connection exists between subnets. Figure 3(b) shows the network seen in Figure 2 with a low inter-transmission probability between subnets where $T(\eta_k, \eta_l) = 0.10$ and transmission probabilities within the subnets in the range $t_{i,j} = [0.85, 1.0]$. When we add these weak interactions between subnets (i.e., relatively low transmission probabilities), we have a game with an noisy AIOS structure where actions in different subnets have only limited effects on the payoffs.

4 Hierarchical Solution Method

We now describe a solution approach that constructs subgames based on strategy clusters and uses the solutions to these subgames to create a more accurate abstracted game. When games have exact AIOS structure, this will result in finding an exact solution to

the original game by composing the results of the subgame solutions. In cases where games have noisy AIOS structure, we propose an iterative solution method that improves solution quality by taking into account error from outside of the subgames.

4.1 Subgames

Consider the AIOS example shown in Figure 1. Ten subgames correspond to ten pairs of clusters of actions for the players. For example, G_1 is played using clusters $c_{1,1}$ and $c_{2,1}$. Now we consider building an abstracted game by first solving each of the subgames G_1 to G_{10} utilizing any solution concept to get a mixed strategy for each player in each game. The abstracted game will have one action for each player corresponding to each cluster (10 in the example). To fill in the payoffs for each pair of clusters (a 10x10 matrix), we compute the expected payoffs using the mixed strategies for the corresponding clusters (for the subgames, this is the expected payoff from the solution to the game). Figure 4 shows the resulting abstracted game R. Next, we solve R using any solution concepts mentioned in section 4.3. To get the reverse mapping here we must distribute the probabilities of $c_{1,1}, c_{1,2}, ..., c_{1,10}$ over all the actions in $c_{1,1}, c_{1,2}, ..., c_{1,10}$ for player 1 to get the strategy for the original game (resp. for player 2). Equation 1 gives this reverse mapping, where $i \in N, \forall a_{i,k} \in c_{i,m}$. In equation 1 the probabilities $\pi^i(a_{i,k})$ on the right-hand side are the mixed strategies for the subgames. We call this approach Subgame Abstraction and Solution Concept (SASC).

	$c_{2,1}$	...	$c_{2,10}$
$c_{1,1}$	$G_{1,1}, G_{1,2}$	...	
.	.		.
.	.	...	.
.	.		.
$c_{1,10}$	...	...	$G_{10,1}, G_{10,2}$

Fig. 4: Abstracted (hierarchical) Game R

$$\pi^i(a_{i,k}) = \pi^i(c_{i,m}) \times \pi^i(a_{i,k}) \tag{1}$$

4.2 Noisy AIOS Games

The AIOS structure is strict if we require identical payoffs outside of the subgames. However, it is much more plausible to find approximate forms of this structure. For example, in Section 3 we saw how low transmission probabilities between subnets lead to an noisy AIOS game. For a noisy version of AIOS, we define the *delta* (δ) parameter to specify how much variation in the payoffs is allowed outside of the subgames. Let $\delta_{i,k}$ be the maximum payoff difference for any pair of actions in cluster k for player i for any strategy of the of the opponent that is not in the same subgame. δ_i is the maximum of $\delta_{i,k}$ for player i, where k can be from 1 to a number of clusters. Equation 2 picks the maximum δ considering all of the clusters and players. Equation 3, where $(i,j) \in N, i \neq j$, calculates δ for one cluster $c_{i,k}$ for a player i.

$$\delta = max_{i \in N}(max(\delta_{i,k})), \qquad k = 1, ..., |\hat{A}_i(R)| \tag{2}$$

$$\delta_{i,k} = max(u_i(a_{i,m}, a_{j,t}) - u_i(a_{i,n}, a_{j,t})) \tag{3}$$

4.3 Solving Games

We consider several solution methods for solving games. We consider both pure and mixed-strategy Nash equilibrium, as well as a different concept that directly minimizes the bound on the approximation quality in the original game.

Approximate Pure Strategy Nash Equilibrium In a Pure-Strategy Nash Equilibrium (PSNE) all players play pure strategies that are mutual best-responses. However, PSNE is not guaranteed to exist. Therefore, we instead look for the pure-strategy outcome that is the best approximate equilibrium. We first calculate the values of deviations for each action a_i and then select the action profile that minimizes the maximum benefit to deviating.

Mixed Strategy Nash Equilibrium We also calculate a version of mixed-strategy Nash equilibrium using the software package Gambit [14]. There are several different solvers for finding Nash equilibria in this toolkit. We used one based on Quantal response equilibrium (QRE) [12].

Minimum Epsilon Bounded Equilibrium When solving an abstracted game, the best analysis may not be finding a Nash Equilibrium, since this may not be an equilibrium of the original game. As an alternative, we introduce *Minimum Epsilon (ε) Bounded* equilibrium (MEB). Instead of considering deviations to clusters of actions (and the average payoff of the cluster), we use the maximum expected payoff for any of the actions in the original game. This heuristic allows for a better estimate of how close the outcome will be to an equilibrium in the original game. The difference in comparison with PSNE is in the calculation of $\varepsilon(a_i^*)$. Equation 4 is used to compute the ε for MEB.

$$\varepsilon(\hat{a}_i^*) = max_{\forall \hat{a}_i \in \hat{A}_i, \hat{a}_j \in \hat{A}_j} [\overline{u}_i(\hat{a}_i, \hat{a}_j) - \hat{u}_i(\hat{a}_i^*, \hat{a}_j)] \tag{4}$$

In the above equation $\overline{u}_i(\hat{a}_i, \hat{a}_j)$ returns a payoff from an upper bound game $\overline{R}$. Payoffs for the upper-bounded game $\overline{R}$ are computed using Equation 5. Equation 5 calculates the maximum expected payoff for an abstracted action by reverse mapping to the original actions and calculating the expected payoff for every original action, selecting the maximum one. Next, where $\forall \hat{a}_i \in \hat{A}_i(R), \forall \hat{a_j} \in \hat{A}_j(R), (i,j) \in N, i \neq j$, the equation iterates over all the actions for every player and calculates the payoffs for the upper-bounded game $\overline{R}$. Equation 4 cannot be used in the original game because we need an upper-bounded game where we use reverse mapping. Unless we have an abstracted game, it is not possible to compute an upper-bounded game.

$$\overline{u_i}(\hat{a_i}, \hat{a_j}) = max_{\forall a_{i,k} \in g(\hat{a}_i)} \frac{\sum_{\forall a_{j,l} \in g(\hat{a}_j)} u_i(a_{i,k}, a_{j,l})}{|g(\hat{a}_j)|} \tag{5}$$

Double Oracle Algorithm The Double Oracle (DO) is not a solution concept. It is a technique used to handle massive games. Double Oracle Algorithms [3] [15] utilize the method of column/constraint generation. The idea is to restrict the strategies of all the

players and solve the restricted game exactly using the LP [16] for solving an NFG. We used the QRE [12] [14] to solve the restricted general-sum NFG. The QRE gives an approximate Nash Equilibrium.

Counter Factual Regret Counterfactual Regret Minimization (CFR) [18] is an iterative algorithm to find approximate Nash Equilibrium. In every iteration, it updates the strategies of the players to minimize a weighted sum of regret at each decision. The average strategies then approach NE.

4.4 Iterative Solution Algorithm

For games with noisy AIOS structure, simply composing (as above) the solutions of the subgames may not be an equilibrium of the original game. The solution may occasionally play in quadrants of the game that are *not* one of the subgames solved explicitly, which results in an error when the payoffs do not match identically. We now introduce an iterative solution technique that (partially) accounts for this error. After solving the subgames and abstracted game as previously, we now calculate the expected payoff for each strategy outside its subgame. Then, we modify the subgames using this error term added to the payoffs in the subgame and solve them again, and then recalculate the abstracted game and solve it again. This process results in a sequence of modified solutions that account for the differences in payoffs outside of the subgames from the previous iteration. We call this algorithm the Iterative Subgame Abstraction and Solution Concept (ISASC).

Consider the subgame G_1 in Figure 1. We want to internalize the noise outside of the subgame into the payoffs of the subgame. So, before solving G_1, we update the payoffs for both player 1 and player 2. For action $\{1-10\} \in c_{1,1}$, we calculate the expected utility when player 2 does not play the actions in the subgame. That means that when player 1 plays $\{1-10\}$, we calculate the expected utility of $\{1-10\}$, denoted Ω_i, by considering the probabilities of player 2 playing $\{11-100\}$ from the strategy on the previous iteration. Then we update the payoffs of G_1 for player 1 for action $\{1-10\} \in c_{1,1}$ for every $\{1-10\} \in c_{2,1}$ by adding the Ω_i. This process repeats for all strategies in the game.

Pseudocode for updating the subgames is shown in Algorithm 1. Subgames are updated using $[u_i(a_i, a_j) = u_i(a_i, a_j)+\Omega_i(a_i)]$, where $\forall a_i \in A_i(G), \forall a_j \in A_j(G), \forall (i,j) \in N, i \neq j$, line 4-12. Lines 5-7 are used to compute the expected payoff Ω_i, for an action of player i, when player j plays outside of the subgame G. The action set for player i in game G is $A_i(G)$. The probability of action a_j from the mixed strategy for iteration $T-1$ is $\pi_{T-1}(a_j)$.

5 Experiments

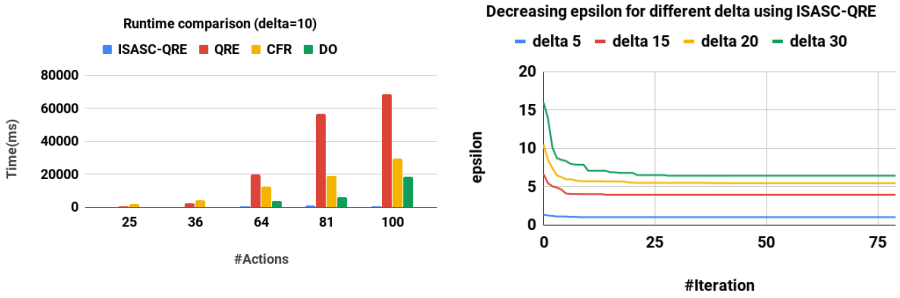
In the experiment section, we used two criteria to measure the performance of our proposed algorithm: (a) runtime (b) epsilon (ϵ). ϵ measures whether there is an incentive for a player to switch to another pure strategy from the current Nash Equilibrium strategy (which can be either a mixed strategy Nash Equilibrium using QRE or a pure strategy

Algorithm 1 Update Subgame Algorithm

Input: Subgame `G'`, original game `G`, player i

```
Subgame actions o' = Actions(G', i)
Opponent actions in G', p = Actions(G, G', i_op)
Opponent actions outside G', p' = OutActions(G, G', i_op)
for j ← 1, o' do                                  ▷ for every action of player i in G'
    for k ← 1, p' do                              ▷ for every action of opponent ∉ G'
        ω = ω + PayOff(j,k,G) ×π(k)
    end for
    for l ← 1, p do                               ▷ for every action of opponent in G'
        Outcome o = [ j, l ]
        G'(o,i) = PayOff(G, o) + ω                ▷ update the payoff in G'
    end for
end for
```

Nash Equilibrium using PSNE, MEB). To compute ϵ of an approximate Nash Equilibrium strategy for player i first we calculate the expected payoff of player i given the approximate Nash Equilibrium strategy of the players. Next, we check whether there is an incentive for player i to switch to a pure strategy from the current approximate Nash Equilibrium strategy (which can be either pure or a mixed strategy Nash Equilibrium). Finally, we take the maximum of all the players' ϵ which gives us the ϵ for an approximate Nash Equilibrium. In a Nash Equilibrium, there is no incentive to switch to a pure strategy for all the participating players ($\epsilon = 0$).



(a) Measuring performance against QRE, CFR and DO which are applied in the original game

(b) Decreasing ϵ for different δ

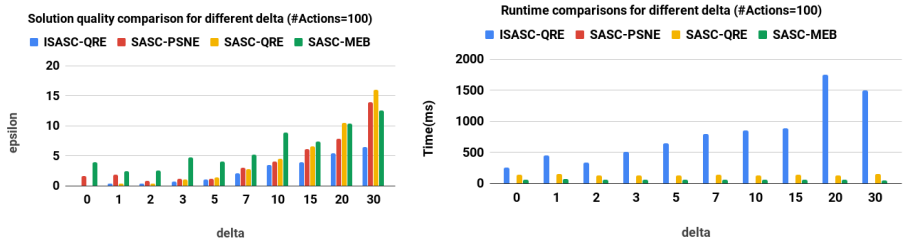
Fig. 5: Measuring performance of ISASC-QRE

For our first experiment we considered 2-player games of different sizes ($\#Actions = 25, 36, 64, 81, 100$) with a fixed $\delta = 10$. For each size, we created 20 different games. For each size, the strategies for each player are partitioned into $5, 6, 8, 9, 10$ clusters with $5, 6, 8, 9, 10$ actions respectively. The subgames are completely random games with payoffs generated uniformly between 0 and 100. The payoffs outside of the subgames are generated randomly with the constraint that in every cluster the maximum

payoff difference between the payoffs for the actions is δ for all actions of the opponent that are not part of the subgame (i.e., for every action outside the subgames we add noise).

We begin by showing that ISASC has benefits when there are limited resources available since ISASC can solve a large game using fewer resources and much more quickly. We compare the runtime performance of ISASC-QRE (ISASC-QRE means we used the ISASC algorithm to solve games where QRE is used to solve the subgames and the hierarchical games) against different solution methods: QRE, CFR and DO, when these different methods are applied to the original game without the use of any abstraction as shown in Figure 5a. The results clearly show that ISASC-QRE was able to solve games faster than QRE, CFR and DO by considerable margins.

Our next experiment focuses on showing that the iteration scheme in ISASC-QRE is effective at improving solution quality. For this experiment we created 20 2-player games for each $\delta = \{5, 15, 20, 30\}$ where 100 actions were available for each player. In Figure 5b we show the error (quantified by the ϵ in the original game) for different levels of δ as we increase the number of iterations. We can see a clear improvement in solution quality with increasing iterations. We also note that the biggest improvements come in the cases with the largest values of δ.



(a) Comparison of solution quality (b) Comparison of runtime

Fig. 6: Performance comparisons for ISASC and SASC algorithms

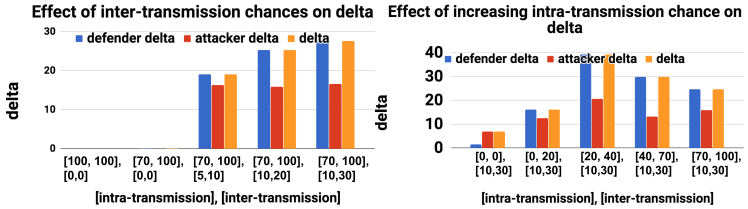
The next experiment compares the solution quality of iterative (ISASC) and non-iterative (SASC) subgame abstraction techniques: ISASC-QRE, SASC-PSNE, SASC-QRE, SASC-MEB. For each algorithm, we explicitly mentioned which solution concept is used to solve the subgames and hierarchical games. For example, in SASC-QRE, we used the QRE to solve the subgames and the hierarchical games. The only exception is SASC-MEB, where we used QRE to solve the subgames since MEB can only be used to solve a hierarchical/abstract game. For this experiment, we created 20 2-player games for each $\delta = \{0, 1, 2, 3, 5, 7, 10, 15, 20, 30\}$ where 100 actions were available for each player. The strategies for each player are partitioned into 10 clusters with 10 actions for each: $|c_1| = |c_2| = 10$, $|c_{1,m}| = 10$, $m = 1, 2, ..., 10$. For this experiment, we assumed that the subgames are known to ISASC and SASC algorithms. Figure 6a 6b shows the results. ISASC-QRE and SASC-QRE does very well in cases with low δ, as expected. However, ISASC-QRE continues to perform better when the values of δ are much more significant. Figure 6a 6b also show that there is a tradeoff between solution

quality and runtime. ISASC-QRE produces the better results but requires more time than SASC-QRE.

Parameter	Value range
$t_{i,j} = t_{j,i}$	$[70, 100]$
$T(\eta_i, \eta_j)$	$[10, 30]$
v_i	$[6, 10]$
$c^d(v_i)$	$[1, 3]$
$c^a(v_i)$	$[1, 3]$
$\lvert e_{\eta_k}, e_{\eta_l} \rvert$	1
$\lvert e_{\eta_k} \rvert$	$\lvert e_{min}, e_{max} \rvert$

Table 1: Network settings

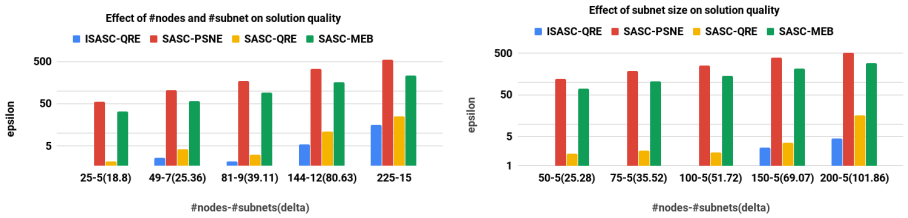
We now consider the more realistic cyber defense games described in Section 3. We compared our ISASC and SASC algorithms. We generated 20 games using the parameter settings shown in Table 1. Each parameter is drawn uniformly from the given range. The number of edges in subnet η_l is $|e_{\eta_l}|$ in the range $|e_{min}, e_{max}|$ where e_{min} and e_{max} are the minimum and maximum number of edges respectively. All networks are connected, and the parameters $T(\eta_k, \eta_l)$ and $t_{i,j}$, where $t_{i,j} >> T(\eta_k, \eta_l)$ control worm propagation. We use Monte Carlo simulation for $10,000$ iterations to estimate the payoffs based on the propagation of the attack. Each subnet forms a cluster of actions for our solution methods. Since we already know the subnets for this cyber defense scenario, and thus the subgames, we assumed that the subgames are already known.



(a) Impact: $T(\eta_k, \eta_l)$ on δ (b) Impact: $t_{i,j}$ on δ

Fig. 7: Effect of transmission parameters on δ

Our first experiment shows how δ varies as we vary the *inter* and *intra-transmission probability* in Figure 7a,7b. We used games with 50 nodes and 5 subnets with the same number of nodes. We can see in Figure 7a that when $t_{i,j} = [100, 100]$ and $T(\eta_k, \eta_l) = 0$ so the subnets are totally disconnected from each other $\delta = 0$. In this case, we can find an exact equilibrium by composing subgame solutions. However, when $T(\eta_k, \eta_l)$ starts to increase δ increases. In both Figure 7a and Figure 7b, we see that δ reaches a maximum when $T(\eta_k, \eta_l) \approx t_{i,j}$ as the spreading of botnet becomes random across the entire network, losing the AIOS structure.



(a) #node and #subnet vs ϵ (b) Subnet size vs ϵ

Fig. 8: Performance of ISASC-QRE

Next, we show how δ and ϵ change when we vary both network size and subnet size. In Figure 8a we can see that ISASC-QRE performs favorably compared to the other solution algorithms. Next, we increase subnet size but keep the number of subnets fixed. Figure 8b shows that as the subnet size increases δ increases. However, the ISASC algorithm continues to provide better solution quality with very low ϵ for higher δ. In all of the experiments, ISASC gives very high solution quality compared to other algorithms.

6 Conclusion

Defending a network against malicious worm requires sophisticated defense mechanism. However, due to large network size and limited resources, it's difficult for a network administrator to harden the security in every host of the network. Solving the game becomes harder due to large action space of the game model. We propose a new class of abstraction methods for NFG based on the AIOS structure. We show that there exist several abstraction-based solution methods that can take advantage of this structure to quickly find solutions to huge games by decomposing them into subgames. For games with only noisy AIOS structure, we show that iterative solution methods can give us very high-quality approximations to the solution of the original game.

Acknowledgement

This research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes not with standing any copyright notation here on.

References

1. Alpcan, T., Basar, T.: An intrusion detection game with limited observations. In: 12th Int. Symp. on Dynamic Games and Applications, Sophia Antipolis, France. vol. 26 (2006)

2. Bard, N., Nicholas, D., Szepesvári, C., Bowling, M.: Decision-theoretic clustering of strategies. In: AAMAS (2015)
3. Bosansky, B., Kiekintveld, C., Lisy, V., Pechoucek, M.: An exact double-oracle algorithm for zero-sum extensive-form games with imperfect information. Journal of Artificial Intelligence Research **51**, 829–866 (2014)
4. Brown, N., Ganzfried, S., Sandholm, T.: Hierarchical abstraction, distributed equilibrium computation, and post-processing, with application to a champion no-limit Texas hold'em agent. Tech. rep. (2014)
5. Conitzer, V., Sandholm, T.: A technique for reducing normal-form games to compute a nash equilibrium. In: AAMAS. pp. 537–544 (2006)
6. Fabrikant, A., Papadimitriou, C., Talwar, K.: The complexity of pure nash equilibria. In: Proceedings of the thirty-sixth annual ACM Symposium on the Theory of Computing. pp. 604–612 (2004)
7. Ganzfried, S., Sandholm, T.: Potential-aware imperfect-recall abstraction with earth mover's distance in imperfect-information games. In: Conference on Artificial Intelligence (AAAI) (2014)
8. Gilpin, A., Sandholm, T.: A competitive texas hold'em poker player via automated abstraction and real-time equilibrium computation. In: Proceedings of the National Conference on Artificial Intelligence (AAAI). vol. 21, p. 1007 (2006)
9. Gilpin, A., Sandholm, T.: Better automated abstraction techniques for imperfect information games, with application to texas hold'em poker. In: AAMAS. p. 192 (2007)
10. Gilpin, A., Sandholm, T., Sørensen, T.B.: Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of texas hold'em poker. In: Proceedings of the Conference on Artificial Intelligence (AAAI). vol. 22, p. 50 (2007)
11. Gilpin, A., Sandholm, T., Sørensen, T.B.: A heads-up no-limit texas hold'em poker player: discretized betting models and automatically generated equilibrium-finding programs. In: AAMAS. pp. 911–918 (2008)
12. Goeree, J.K., Holt, C.A., Palfrey, T.R.: Quantal response equilibrium. The New Palgrave Dictionary of Economics. Palgrave Macmillan, Basingstoke (2008)
13. Matta, V., Di Mauro, M., Longo, M.: Ddos attacks with randomized traffic innovation: botnet identification challenges and strategies. IEEE Transactions on Information Forensics and Security **12**(8), 1844–1859 (2017)
14. McKelvey, R.D., McLennan, A.M., Turocy, T.L.: Gambit: Software tools for game theory (2006)
15. McMahan, H.B., Gordon, G.J., Blum, A.: Planning in the presence of cost functions controlled by an adversary. In: Proceedings of the 20th International Conference on Machine Learning (ICML-03). pp. 536–543 (2003)
16. Shoham, Y., Leyton-Brown, K.: Multiagent systems: Algorithmic, game-theoretic, and logical foundations. Cambridge University Press (2008)
17. Venkatesan, S., Albanese, M., Shah, A., Ganesan, R., Jajodia, S.: Detecting stealthy botnets in a resource-constrained environment using reinforcement learning. In: Proceedings of the 2017 Workshop on Moving Target Defense. pp. 75–85. ACM (2017)
18. Zinkevich, M., Johanson, M., Bowling, M., Piccione, C.: Regret minimization in games with incomplete information. In: Advances in neural information processing systems. pp. 1729–1736 (2008)