Investigating algorithms for finding Nash equilibria in cyber security problems

Linda Zhang
MIT, CSAIL
Cambridge, Massachusetts
lolzhang@mit.edu

Erik Hemberg
MIT, CSAIL
Cambridge, Massachusetts
hembergerik@csail.mit.edu

ABSTRACT

Distributed Denial of Service (DDoS) cyber attacks continue to increase and cause disruptions in both industry and politics. As more critical information and services are provided through networks, it is important to keep these networks available. However, since adversaries are continuously changing and adapting, stationary defense strategies do not effectively secure networks against attacks. We investigate Nash equilibria in cyber security problems by modeling attacker-defender interactions using competitive coevolutionary algorithms. In particular, we examined the performances of two algorithms (and their variations) that look for Nash equilibria, `NashSolve` and `HybridCoev`, and compared their performances against other existing heuristics. Using two evaluation techniques, one that looked at average fitness scores and one that created a compendium of MEU, MinMax, and inverse Pareto front ratio scores, we found that `NashSolve` and `HybridCoev` did not perform significantly better for both attacker and defender populations relative to other heuristics.

KEYWORDS

cyber security, co-evolution, network, evolutionary algorithms

ACM Reference Format:
Linda Zhang and Erik Hemberg. 2019. Investigating algorithms for finding Nash equilibria in cyber security problems. In *GECCO '19 Companion: Genetic and Evolutionary Computation Conference Companion, July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3205651.3208287

1 INTRODUCTION

Cyber attacks have become increasingly sophisticated, dangerous, and frequent as more information and technologies move online. Although systems can be secured against attacks, adversaries may learn from these defenses and find new ways to breach them. Defenders must then adapt to these new attacks and determine new strategies, and the cycle repeats. Therefore, cyber adversaries is a perpetual security problem. When considering networks, resources may not be dynamic enough to constantly reroute and change; thus, we want to be able to select the best defense solution that is robust against autonomous and adaptive adversaries [18].

One way to evaluate solutions in a multi agent setting is to consider Nash equilibria. These are points which satisfy every player's optimizing condition given the other players' choices. That is, a player does not have incentive to deviate from its strategy given the other players' strategies. This concept has been used to understand the strategic actions of multiple players in a deterministic gaming environment [16]. Since we can model different network threat scenarios as games, Nash equilibria may offer insight into attacker-defender coevolution.

Our research questions focus on evaluating Nash equilibria in the context of attacker-defender coevolutionary algorithms. How do the performances of two Nash equilibrium finding algorithms, `NashSolve` [23] and `HybridCoev` [27], perform compared against existing and previously tested heuristics? How do we know when we have found a Nash equilibrium? Can we find ways to further improve these algorithms?

The contributions of this paper are:

(1) Evaluating coevolutionary algorithms that look for Nash equilibriums in the context of cyber security problems.
(2) Introducing variants of competitive coevolutionary Nash equilibrium finding algorithms.
(3) Objectively comparing performance of Nash equilibrium-finding algorithms against existing coevolutionary algorithms.

The Background section (Section 2) gives an overview of previous and related work, including Nash equilibriums, cyber security, and Nash equilibriums in the context of cyber security. The Method section (Section 3) describes coevolutionary algorithms, in particular, two Nash equilibrium-finding algorithms: `NashSolve` and `HybridCoev`. The Experiment section (Section 4) details the setup, including the Mobile Asset Placement problem, settings, and hold-out sets, as well as results and discussion. Finally, the Conclusion section (Section 5) summarizes our findings and lists potential areas for future work.

2 BACKGROUND

This section defines a Nash equilibrium, provides background on competitive coevolution, as well as the use of competitive coevolution in Cyber Security.

2.1 Nash equilibrium

A strategy is a Nash equilibrium if no player can improve by unilaterally changing their strategy. Let (S, f) be a game with n players, where S_i is the strategy set for player i, $S = \{S_1 \times S_2 \times \cdots \times S_n\}$ is the set of strategies and $f(x) = (f_1(x), \ldots, f_n(x))$ is the payoff function,

GECCO '19 Companion, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-5764-7/18/07…$15.00
https://doi.org/10.1145/3205651.3208287

$x \in S$. Let x_i be a strategy of player i and x_{-i} be a strategy of all players except player i. When each player $i \in \{1, \ldots, n\}$ chooses strategy x_i giving the strategies $x = (x_1, \ldots, x_n)$ then player i obtains payoff $f_i(x)$. The chosen strategies $x^* \in S$ is a Nash equilibrium (NE) if no unilateral deviation in strategy by any single player is profitable for that player, that is

$$\forall i, x_i \in S_i : f_i(x_i^*, x_{-i}^*) \geq f_i(x_i, x_{-i}^*).$$

When the inequality above holds strictly, then the equilibrium is called a "strict Nash equilibrium". If there is a player with an exact equality between x_i^* and another strategy in the set S, then the equilibrium is called a "weak Nash equilibrium".

Nash equilibria can be difficult to find with populations. In the context of populations it is the evolutionary stable strategies that are of interest. An evolutionary stable strategy occurs when the whole population is using this strategy and any small group of invaders using a different strategy will eventually die off over multiple generations. If and only if a strategy is evolutionary stable it is a Nash equilibria, not the reverse.

2.2 Nash Equilibrium and Evolutionary Computation

A basic competitive coevolutionary algorithm evolves two coupled populations, each with selection and variation (crossover and mutation). One population comprises *tests* and the other *solutions* [20]. In each generation, different competitions are formed by pairing a test and a solution. This couples the two population as they share a fitness evaluation component.

A test competes to demonstrate the solution as incorrect. The solution competes to solve the test correctly. The dynamic of the algorithm, driven by conflicting objectives and guided by performance-based selection and random variation can gradually produce better and more robust solutions (i.e defenses) [21, 26]. Competitive coevolutionary algorithms are often applied in domains in which there is no exogenous objective measure of performance but wherein performance is relative to others is a good measure. These have been called *interactive* domains in [11, 20] and include games, e.g. hybrid coevolutionary programming for Nash equilibrium search in games with local optima [27].

A fitness score is derived from some function of its performance outcomes in its competitions. Methods have been defined depending on the specific problem domain, e.g., [1, 4, 6, 12, 28]. A more formal approach, using the *solution* and *test* perspective, describes fitness assignments as *solution concepts* [20]. Solution concepts include: best worst case, maximization of expected utility, Nash equilibrium, and Pareto optimality. One of the key challenges with coevolution is how to train and rate the performance of solutions [2, 3].

2.3 Cyber security and Competitive Coevolution

Network defense is studied with the coevolutionary agent-based network defense lightweight event system (CANDLES) [22] a framework designed to coevolve attacker and defender agent strategies with a custom, abstract computer network defense simulation. The RIVALS network security framework, supports three studies into, respectively DDOS, deceptive and isolation defense [18]. Malware coevolution was studied for mobile malware and anti-malware [25]. Other examples of competitive coevolution in cyber security can be found in [8, 9, 15, 17].

There is scope to expand the work regarding finding Nash equilibrium solutions with competitive coevolutionary algorithms in cyber security. This paper will further investigate this.

3 METHOD

Competitive coevolutionary algorithms can be used to model the "arms race" between attackers and defenders. In particular, a previous project RIVALS [7] is used to determine the best defense strategy for a network, specifically using peer-to-peer network simulation and simulated extreme distributed denial of service (DDoS) attacks.

Using competitive coevolutionary algorithms, populations iteratively evolve against each other by evaluating the fitness of solutions against adversaries. The fitness values assigned to individuals at a generation are relative to the adversaries that the individuals were evaluated against, and these fitness evaluations are the most computationally cost intensive portion of the algorithm. Individuals with the highest fitness are selected in each population, and their offspring then produce the next generation of populations. These offspring inherit portions of their parents' genomes through a specified crossover probability, and further variation is introduced by randomly changing the genome through a specified mutation probability. The process is then repeated for a specified number of epochs or until convergence.

In this section we present the methods we will use to find Nash equilibria. We look at two algorithms, `NashSolve` and `HybridCoev`, that attempt to find Nash equilibria within a problem.

3.1 NashSolve

The `NashSolve` algorithm uses an archive to try and solve the problem of cycling in the solution space, which leads to suboptimal or mediocre solutions [23]. This occurs due to intransitivities, where there are cycles within the "A beats B" relation on a set of players.

`NashSolve` starts with a random player in a fixed-size archive of players. We compare all of our candidate solutions with this player, and record their performance. If the worst case performance of a new solution against the archive is above a specified lower threshold, we add it to the archive. We then iterate, assigning to each candidate solution its worst score against all the players in the archive. When the size limit is reached, any new player added to the archive replaces the oldest player there. Traditional coevolution is then performed on the populations.

Additionally, we explored two variations on the original `NashSolve` algorithm. In `NashBestSolve`, we compare best case performance of a new solution against the archive, and in `NashAvgSolve`, we compare average case performance of a new solution against the archive.

3.2 HybridCoev

The `HybridCoev` algorithm tries to overcome the problem of "local Nash equilibrium traps" [27]. These are points that follow a local optimization path but are not true global Nash equilibrium points.

`HybridCoev` players have a set of multiple strategies, where each strategy has its own score. The fitness of a `HybridCoev` individual is determined to be the best score of its strategies. Additionally, we evaluate the scores of a strategy population by randomly

Algorithm 1 NashSolve

```
1: define global input Evolutionary Algorithm (EA), maxArchiveSize, lowerThreshold
2: procedure NashSolve
3:     for i ← 0 to generations do
4:         Archive ← {RandomPlayer}
5:         Solutions ← EA.getCandidateSolutions()
6:         for each s in Solutions do
7:             minScore ← ∞
8:             for each a in Archive do
9:                 score ← eval(s, a)
10:                if score < minScore then
11:                    score ← minScore
12:            s.score ← minScore
13:        sort(Solutions)
14:        if Solutions[0] < lowerThreshold then
15:            Archive.add(Solutions[0])
16:            if |Archive| > maxArchiveSize then
17:                Archive.remove(0)
18:        EA.pushSolutions(Solutions)
19:        EA.evolve()
       return Archive[−1]
```

choosing a strategy from a random player in the population and then playing it against a randomly chosen strategy from a random player in the rival population until all strategies have been played.

`HybridCoev` starts with a randomly chosen strategy of a player in a population. We then take the best strategies of all other rivals from the previous generation and apply a local hill climber for fine tuning of the chosen strategy of the player. This chosen strategy is then replaced by the new finely tuned strategy. These steps are repeated as many times as set by the best rival matching rate, and then the process is repeated for every other player in turn. Traditional coevolution is then performed on the populations.

4 EXPERIMENTS

In this section we present our empirical investigation into on evaluating Nash equilibria in the context of attacker-defender coevolutionary algorithms. How do the performances of Nash equilibrium finding algorithms perform compared against other competitive coevolutionary heuristics? How do we know when we have found a Nash equilibrium? Our hypothesis is that it will be difficult for the competitive coevolutionary algorithms to find Nash equilibria, even for the heuristics that claim to be designed to find them, due to the added complexities of population based search.

First we present the setup of the problem and heuristics, and then the results.

4.1 Setup

4.1.1 Mobile Asset Placement Problem. The Mobile Asset Placement (MAP) problem is based on the problem described in RIVALS [7]. MAP illustrates the worst-case scenario in a network. A network runs the Chord protocol, and a mission is simulated where an attacker will take out a set of nodes for the entirety of the mission while a defender attempts to complete a set of tasks consisting of a start node and an end node. A task succeeds as long as both nodes in the task are not taken out by the attacker, and the mission succeeds if all tasks succeed. Note that MAP models DOS threats and looks at where to allocate resources, so the problem determines quality of service, not detection. Additionally, MAP is static, so we can quickly evaluate strategies while considering a worst-case vulnerability. A sample grammar for MAP topology 1 (Figure 1) is given below.

```
<attacker> ::= [<attacks>]
<attacks> ::= <node>, <attacks> | <node>
<node> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6
<defender> ::=
[task(<node>, <node>), task(<node>, <node>), task(<node>, <node>),
 task(<node>, <node>), task(<node>, <node>), task(<node>, <node>)]
<node> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6
```

We ran experiments on three different MAP topologies, with variations shown in Table 1. Topology 1 is a small network used for testing and verification of algorithm correctness. Topologies 2 and 4 are medium-sized networks with very different edge connections that are more reflective of security problems. Illustrations of these topologies are shown in Figures 1, 2, and 3.

Table 1: MAP topology variations

Topology	# Nodes	# Tasks
1	7	6
2	23	4
4	16	5

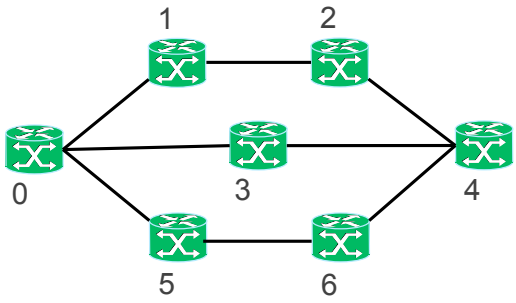

Figure 1: MAP topology 1

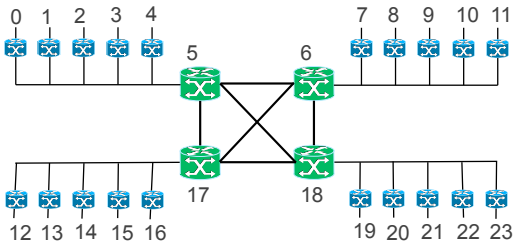

Figure 2: MAP topology 2

In order to measure how an individual performs against an adversary, we use the same fitness functions previously defined by RIVALS for network placement.

Algorithm 2 HybridCoev

```
define global input Evolutionary Algorithm (EA), adversaryMatchingRate
procedure HYBRIDCOEV
    for each population in populations do
        for each individual in population do
            individual.strategies ← {RandomStrategies}
    for i ← 0 to generations do
        eval_all_strategies()
        EA.evolve()
        for individual in population do
            for s in adversaryMatchingRate * individual.strategies do
                bestAdversaryStrategies ← individual.adversary.bestStrategies
                tunedStrategy ← hillclimber(s, bestAdversaryStrategies)
                s ← tunedStrategy
        EA.pushSolutions(population)
    return best strategies
procedure EVAL_ALL_STRATEGIES
    for each population in populations do
        randomIndividuals ← randomize(population.individuals)
        while randomIndividuals do
            individual ← randomIndividuals.pop()
            s1 ← random(individual.strategies)
            s2 ← random(random(individual.adversary).strategies)
            s1.score ← eval(s1, s2)
```

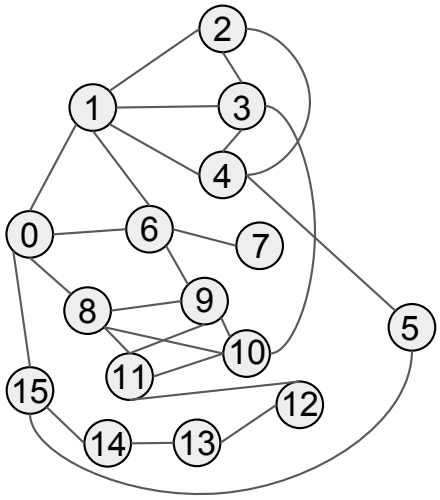

Figure 3: MAP topology 4

The attacker fitness is calculated by

$$f_a = \frac{n_{failed}}{n_{tasks}} - \frac{n_{attacks}}{c \cdot n_{tasks}}$$

where n_{tasks} is the total number of tasks, n_{failed} is the number of tasks the attacker was able to disrupt, $n_{attacks}$ is the number of attacks the attacker used, and c is some constant. This formula helps to incentive attackers to disrupt tasks using as few attacks as possible.

The defender fitness is calculated by

$$f_d = \frac{n_{successful}}{n_{tasks}} - n_{same\_nodes} - n_{duplicate\_tasks}$$

where $n_{successful}$ is the total successful tasks, n_{same_nodes} is the number of tasks with the same start and end node, and $n_{duplicate\_tasks}$ is the number of duplicated tasks. This function helps incentivize defenders to succeed at as many tasks as possible while penalizing approaches that use trivial tasks (same start and end node) or duplicate tasks.

4.1.2 Settings. We want to find an optimal heuristic of obtaining an accurate solution while still being efficient with time and other resources. The most computationally costly element of a coevolutionary algorithm is the fitness evaluation, and the number of fitness evaluations done is proportional to population size and number of epochs/generations. Thus, we ran experiments varying these parameters with ratios to 100 to determine performance differences between heuristics. We ran 30 trials of the following `(population_size` p`, generations` g`)` pairs: $(p = 2, g = 50)$, $(p = 5, g = 20)$, $(p = 10, g = 10)$, $(p = 20, g = 5)$.

Settings used for all algorithms and heuristics are as follows:

- `tournament_size=2`
- `crossover_probability=0.8`
- `mutation_probability=0.1`

Additionally, experiment-specific settings for the algorithms evaluated are shown in Table 2.

4.1.3 Hold-out Set. For each algorithm, we calculate the fitness of the best solutions at each generation averaged over all runs. However, solution fitness is relative to the fitness of the adversary

Table 2: Competitive coevolutionary algorithm-specific settings

Algorithm	Settings
`DiscoCoev` [13]	`n_clusters=3`
`DofCoev` [14]	`dof_impute_strategy=mean, dof_alpha=.8, dof_num_components=2`
`GaupRecCoev` [19]	`gauprec_alpha=.25`
`GridCoev` [19]	`host_population=defender, host_solution_concept=meu`
`HybridCoev` [27]	`num_strategies=3, rival_matching_rate=.9, max_hill_climber_iter=5`
`IPCACoev` [10]	`win_threshold=.1, parent_archive_probability=.9`
`MaxSolveCoev` [5]	`archive_size=20`
`MEULockstepCoev` [19]	`locked_population=defender, locked_population_generations=2, locked_population_elite_size=1, locked_population_parents_size=2`
`MinMaxCoev` [7]	N/A
`MinMaxLockstepCoev` [19]	`locked_population=defender, locked_population_generations=2, locked_population_elite_size=1, locked_population_parents_size=2`
`MuleGE` [7]	`mule_ge_elite_size=1, one_way_population=attacker`
`NashSolve` [23]	`lower_threshold=-2, max_archive_size=3`
`RIPCACoev` [7]	`win_threshold=.1, parent_archive_probability=.9`
`SimpleCoev` [7]	N/A

population at that generation during the run. Additionally, different heuristics may have different ways of computing fitness. Thus, the fitness scores calculated during the experiments are not absolute, and algorithm performances cannot be objectively compared in this way.

In order to objectively compare the performances of different algorithms, for each experiment we evaluate each algorithm's best solution at each generation against a randomly generated "hold-out set" of attackers and defenders. We then compare the fitness of solutions against this hold-out set to have a constant standard for performance.

4.2 Results

A sample of high performing attacker and defender solution are shown in Figures 4 and 5.

We will primarily show results from MAP topology 2 with ($p = 10, g = 10$) and MAP topology 4 with ($p = 20, g = 5$). We believe that this data is a good representation of the general performance of these algorithms with average population sizes and generations on larger, more relevant topologies.

We will first look at `NashSolve` and `HybridCoev` performance individually, then compare them with existing algorithms.

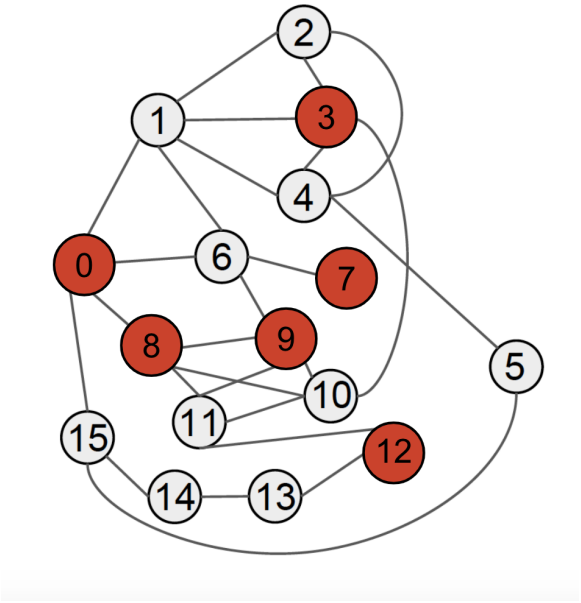

Figure 4: High performing attacker on MAP topology 4 with ($p = 20, g = 5$), red marks the attacked nodes.

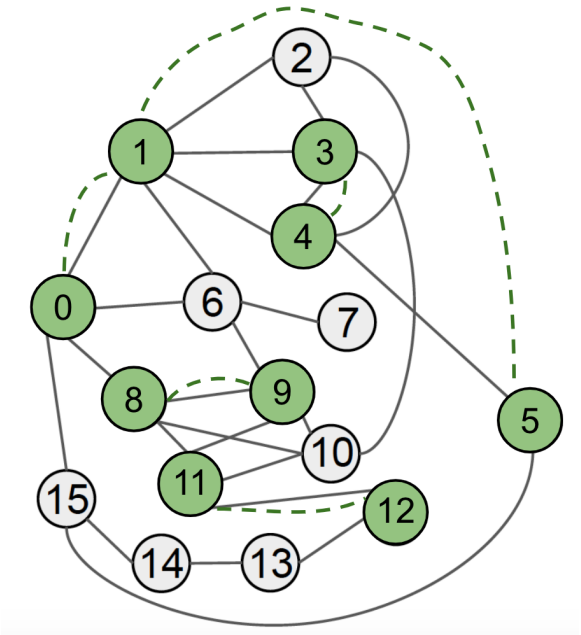

Figure 5: High performing defender on MAP topology 4 with ($p = 20, g = 5$), green marks the tasks and the dashed line the connections.

The **best**, *median*, and worst scores are marked in all the result tables.

4.2.1 NashSolve. Tables 3 and 4 show results for the `NashSolve` variants for various MAP topologies and `(population_size, generations)` pairs. The values shown are the fitness scores of the best final generation solutions averaged across all runs. Additionally, Figure 6 compares variation performance against the holdout set over time. We see that there was not much variance in attacker or defender performance within `NashSolve` variants for a given MAP topologies and `(population_size, generations)` pair. That is, comparing worst (`Nash`), average (`NashAvg`), and best fitness (`NashBest`) solutions against the archive did not seem to significantly affect performance.

Table 3: `(population_size, generations)` pairs for`NashSolve` attacker performance final fitness against hold-out set.

Attacker	Nash	NashAvg	NashBest
Topology 2			
(5, 20)	**.948 ± .028**	*.942 ± .023*	.939 ± .020
(10, 10)	.787 ± .028	**.800 ± .030**	*.792 ± .027*
(20, 5)	*.927 ± .012*	.926 ± .009	**.931 ± .009**
Topology 4			
(5, 20)	*.884 ± .046*	**.890 ± .050**	.880 ± .036
(10, 10)	*.759 ± .050*	**.771 ± .049**	.758 ± .052
(20, 5)	.857 ± .022	*.861 ± .030*	**.868 ± .032**

Table 4: `(population_size, generations)` pairs for `NashSolve` defender performance final fitness against hold-out set

Defender	Nash	NashAvg	NashBest
Topology 2			
(5, 20)	.064 ± .111	*.069 ± .148*	**.156 ± .179**
(10, 10)	.106 ± .163	*.157 ± .188*	**.173 ± .204**
(20, 5)	.151 ± .178	**.209 ± .185**	*.166 ± .147*
Topology 4			
(5, 20)	**.091 ± .190**	*.057 ± .137*	.031± .174
(10, 10)	.117 ± .105	**.145 ± .083**	*.121 ± .082*
(20, 5)	*.190 ± .128*	**.192 ± .098**	.169 ± .116

We also find that defender scores are much lower than attacker scores. This is not surprising, as similar results have been found in other heuristics as well. When we look at the solutions themselves, we find that the final generation solutions are not similar for either attacker or defender, and there were no duplicate solutions. This could imply a difficulty in finding a nash equilibrium for the MAP problem.

4.2.2 HybridCoev. When we evaluate the performance of `HybridCoev`, we must consider how to calculate a player's fitness against the hold-out set given that each player has multiple strategies. We use the same `eval_all_strategies()` evaluation to calculate scores for each strategy, but we have determined three ways to calculate player fitness to compare against other algorithm performances: we can either assign a player's fitness as its best strategy score, worst

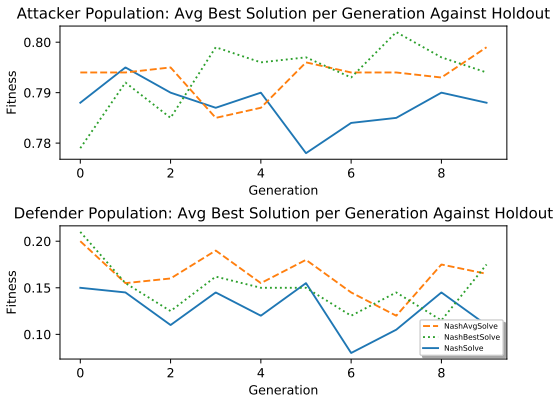

Figure 6: `NashSolve` variations' performances against the holdout set over time on MAP topology 2 with `(population_size=10, generations=10)`, averaged over 30 runs

strategy score, or average of all its strategy scores. Note that the `HybridCoev` algorithm and solution does not change; we only vary the calculation for hold-out set fitness evaluation (columns in Tables 5 and 6).

Table 5 and 6 shows results for the `HybridCoev` fitness calculation variants for various MAP topologies and `(population_size, generations)` pairs. Again, the values shown are the fitness scores of the best final generation solutions averaged across all runs. Upon visual inspection of the tables, and of Figure 7, it looks as if there are differences in fitness scores for calculating using best, average, and worst strategy scores. However, these differences fall within approximately one standard deviation from the average, which may not be as significant. This suggests a variety of strategies per player, and that a given player's strategies may not all converge to one similar strategy.

Table 5: `(population_size, generations)` pairs for `HybridCoev` attacker performance final fitness against hold-out set

Attacker	Worst	Avg	Best
Topology 2			
(5, 20)	.888 ± .157	*.933 ± .082*	**1.00 ± 0.00**
(10, 10)	.633 ± .277	*.777 ± .189*	**.911 ± .191**
(20, 5)	.911 ± .147	*.955 ± .085*	**.977 ± .083**
Topology 4			
(5, 20)	.793 ± .131	*.873 ± .085*	**.986 ± .050**
(10, 10)	.673 ± .167	*.770 ± .142*	**.840 ± .150**
(20, 5)	.760 ± .203	*.830 ± .119*	**.960 ± .080**

Table 6: (population_size, generations) pairs for HybridCoev defender performance final fitness against hold-out set

Defender	Worst	Avg	Best
Topology 2			
(5, 20)	-.322 ± .370	*-.067 ± .200*	**.144 ± .186**
(10, 10)	-.433 ± .405	*-.061 ± .199*	**.289 ± .187**
(20, 5)	-.356 ± .412	*.011 ± .223*	**.356 ± .191**
Topology 4			
(5, 20)	-.587 ± .609	*-.210 ± .329*	**.120 ± .160**
(10, 10)	-.667 ± .542	*-.250 ± .260*	**.220 ± .181**
(20, 5)	-.793 ± .749	*-.257 ± .370*	**.287 ± .184**

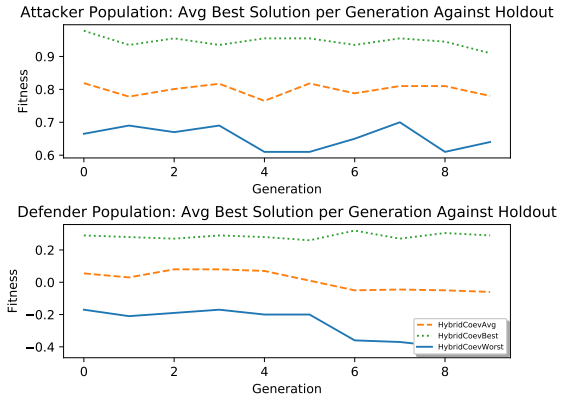

Figure 7: HybridCoev score performance variations against the hold-out set over time on MAP topology 2 with (population_size=10, generations=10), averaged over 30 runs

We again find that defender scores are much lower than attacker scores. Additionally, when we look at the solutions themselves, we find that the final generation solutions are not similar for either attacker or defender. Again, this could imply a difficulty in finding a Nash equilibrium for the MAP problem.

4.2.3 Comparison. To compare NashSolve and HybridCoev performance against other algorithms, we consider multiple evaluation techniques. One method looks at average and final best fitness scores averaged across all runs. The other method creates a compendium [24] by calculating MEU, MinMax, and inverse Pareto front ratio scores to create an objective combined score for a single best solution.

We first look at the average and final fitness scores averaged across all runs. The best individuals per generation are saved and evaluated against the hold-out set, and their fitness scores are recorded for each trial run. The "final" fitness score for an population is the best final generation fitness averaged across all trials, and the "average" fitness score for a population is the average of the best fitness scores across all generations averaged across all trials. Table 7 displays average and final fitness scores for the attacker and defender populations for various algorithms.

When looking at MAP defender solutions ranked by fitness against the hold-out set, we saw that IPCACoev, rIPCACoev, and Minmax-LockstepCoev tended to produce the strongest solutions for all topologies and for all (population_size, generations) pairs, including the example shown in Table 7. We expected IPCACoev and rIPCACoev to perform well for the defender population based on previous empirical data; however, these algorithms have long running time and require more evaluation steps. The NashSolve variations and HybridCoev algorithms did not produce particularly good solutions on average compared to other algorithms. Additionally, note that some NashSolve and HybridCoev final scores are less then the average scores. This implies that the final best solutions did not necessarily perform better than previous generation solutions.

We then compare these heuristics against each other by creating a compendium. Here, we look at each algorithm's performance against the hold-out set using three different solution concepts: MEU is the maximized expected utility, MinMax is the best worst case solution using the fittest test, and inverse Pareto front measures the ratio of the Pareto front number of a solution to the total number of Pareto fronts. We then use these three solution concepts to calculate a combination score based on the summed normalized MEU, MinMax, and inverse Pareto front ratio. In the event of a tie, each ranking scheme first uses MEU score, then MinMax score, then inverse Pareto front ratio to break the tie. If multiple solutions are still tied after this, the top solution is chosen at random from among the tied solutions. Table 8 displays the combined score, as well as the score for each different solution concept, for the highest combined ranking individual solution produced by each algorithm.

When looking at this compendium, we notice that the algorithms rank similarly, particularly for defender solutions. However, for some attacker solutions, the corresponding algorithms that achieve these high combined scores did not perform well in the previous Table 7 comparison. The Nash equilibrium algorithm NashBestSolve and RIPCACoev are prime examples of this. However, the Table 8 results do not necessarily contradict the results from Table 7. The compendium considers the highest combined score for an individual solution rather than averaging over all runs, so algorithms that perform well in Table 8 but do not perform well in Table 7 may be good at finding strong individual solutions but not strong solutions on average.

5 CONCLUSION & FUTURE WORK

We looked at two Nash equilibrium finding algorithms, NashSolve and HybridCoev, and applied them to cyber security problems. We then analyzed their performances individually and compared against other existing algorithms. When considering different evaluation techniques, we found that NashSolve and HybridCoev do not produce particularly strong defense solutions on average, they are able to produce strong individual solutions. As expected it was difficult to identify nash equilibria for MAP with the competitive coevolutionary heuristics, even the ones that claimed to specialize in finding nash equilibria.

Table 7: Algorithm average and final fitness scores against hold-out sets on MAP topology 2 with `(population_size=10, generations=10)`, averaged over 30 runs

	Attacker		Defender	
Algorithm	Avg	Final	Avg	Final
DiscoCoev	.822 ± .005	.830 ± .035	.284 ± .071	.400 ± .195
DofCoev	.809 ± .005	.807 ± .037	.218 ± .043	.177 ± .282
GaupRecCoev	.807 ± .003	*.806* ± *.046*	-.040 ± .062	-.011 ± .328
GridCoev	.802 ± .007	*.806* ± *.032*	*.267* ± *.053*	.327 ± .191
HybridCoevAvg	.796 ± .018	.777 ± .189	.011 ± .057	-.061 ± .199
HybridCoevBest	**.945 ± .018**	**.911 ± .191**	.283 ± .018	*.289* ± *.187*
HybridCoevWorst	.652 ± .033	.633 ± .277	-.272 ± .104	-.433 ± .405
IPCACoev	.815 ± .003	.818 ± .040	.413 ± .076	**.494 ± .227**
MaxSolveCoev	.822 ± .013	.830 ± .047	.337 ± .087	.387 ± .215
MEULockstepCoev	.805 ± .008	.803 ± .045	.324 ± .054	.320 ± .205
MinMaxCoev	.790 ± .006	.783 ± .030	.148 ± .039	.093 ± .153
MinMaxLockstepCoev	.801 ± .005	.801 ± .043	**.423 ± .106**	.479 ± .265
NashAvgSolve	.793 ± .004	.800 ± .030	.162 ± .021	.157 ± .188
NashBestSolve	.792 ± .006	.792 ± .027	.148 ± .028	.173 ± .204
NashSolve	.787 ± .004	.787 ± .028	.125 ± .025	.106 ± .163
RIPCACoev	*.803* ± *.002*	*.806* ± *.036*	.337 ± .069	.422 ± .220
SimpleCoev	.811 ± .005	*.806* ± *.030*	.286 ± .069	.363 ± .185

Table 8: Compendium rankings on MAP topology 2 with `(population_size=10, generations=10)`

	Attacker				Defender			
Algorithm	Coev MEU	Coev MinMax	Inverse Pareto Front	Combined Score	Coev MEU	Coev MinMax	Inverse Pareto Front	Combined Score
DiscoCoev	.897	.330	21	1.338	**.867**	*.333*	*130*	2.200
DofCoev	**.932**	.665	19	1.668	.467	0	**422**	1.467
GaupRecCoev	*.930*	*.663*	11	1.693	.500	0	212	1.500
GridCoev	.898	.322	7	1.282	*.600*	*.333*	126	*1.933*
HybridCoev	.898	.332	1	1.238	.567	*.333*	211	1.900
IPCACoev	*.930*	*.663*	14	1.718	**.867**	*.333*	12	2.200
MaxSolveCoev	.928	.662	3	1.652	.467	0	56	0.967
MEULockstepCoev	*.930*	*.663*	7	1.638	.883	*.333*	128	2.167
MinMaxCoev	**.932**	.665	**22**	1.688	.633	*.333*	221	1.967
MinMaxLockstepCoev	*.930*	*.663*	13	*1.664*	.883	**.667**	61	**2.500**
NashAvgSolve	.899	.332	5	1.281	.533	0	238	1.533
NashBestSolve	**.932**	**.666**	**22**	**1.798**	.533	0	222	1.533
NashSolve	.897	.331	3	1.259	*.600*	*.333*	216	*1.933*
RIPCACoev	*.930*	*.663*	*12*	1.793	*.600*	*.333*	7	*1.933*
SimpleCoev	*.930*	.664	16	1.685	.500	*.333*	40	1.083

For future work, further comparison of different heuristics can be done, for example, multidimensional Elo rating and Nash averaging. Because there is no set objective way to evaluate algorithm performance, different techniques may yield more insight into the algorithm results. Additionally, we would like to look at more scenarios and network simulations other than MAP. If we were to consider non-static problems, we could consider measurements such as latency, bandwidth, r-score, and other network measurements. We could also try to use algorithms for creating classifiers, where we could look at detection and false positives. Finally, we want to consider applying the `NashSolve` and `HybridCoev` algorithms to other problems that have known Nash equilibriums and analyze performance there.

REFERENCES

[1] Peter J. Angeline and Jordan B. Pollack. 1993. Competitive environments evolve better solutions for complex tasks. In *Proceedings of the Fifth International Conference (GA93), Genetic Algorithms*. 264–270.

[2] Kai Arulkumaran, Antoine Cully, and Julian Togelius. 2019. AlphaStar: An Evolutionary Computation Perspective. *arXiv preprint arXiv:1902.01724* (2019).

[3] David Balduzzi, Karl Tuyls, Julien Perolat, and Thore Graepel. 2018. Re-evaluating evaluation. In *Advances in Neural Information Processing Systems*. 3272–3283.

[4] A. B. Cardona, J. Togelius, and M. J. Nelson. 2013. Competitive coevolution in Ms. Pac-Man. In *2013 IEEE Congress on Evolutionary Computation*. 1403–1410.

[5] Edwin De Jong. 2005. The maxsolve algorithm for coevolution. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. ACM, 483–489.

[6] D Fogel. 2001. Blondie24: Playing at the Edge of Artificial Intelligence. (2001).

[7] Dennis Garcia, Anthony Erb Lugo, Erik Hemberg, and Una-May O'Reilly. 2017. Investigating coevolutionary archive based genetic algorithms on cyber defense networks. *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (2017).

[8] Erik Hemberg, Joseph R Zipkin, Richard W Skowyra, Neal Wagner, and Una-May O'Reilly. 2018. Adversarial co-evolution of attack and defense in a segmented computer network environment. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 1648–1655.

[9] P. Hingston and M. Preuss. 2011. Red teaming with coevolution. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*. 1155–1163. https://doi.org/10.1109/CEC.2011.5949747

[10] Edwin D. de Jong. 2007. A Monotonic Archive for Pareto-Coevolution. *Evol. Comput.* 15, 1 (March 2007), 61–93. https://doi.org/10.1162/evco.2007.15.1.61

[11] Krzysztof Krawiec and Malcolm Heywood. 2018. Solving complex problems with coevolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 880–906.

[12] Chong-U Lim, Robin Baumgarten, and Simon Colton. 2010. Evolving behaviour trees for the commercial game DEFCON. In *European Conference on the Applications of Evolutionary Computation*. Springer, 100–110.

[13] Pawel Liskowski and Krzysztof Krawiec. 2014. Discovery of implicit objectives by compression of interaction matrix in test-based problems. In *International Conference on Parallel Problem Solving from Nature*. Springer, 611–620.

[14] Pawel Liskowski and Krzysztof Krawiec. 2016. Non-negative Matrix Factorization for Unsupervised Derivation of Search Objectives in Genetic Programming. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. ACM, 749–756.

[15] Mary L McDonald and Stephen C Upton. 2005. Investigating the dynamics of competition: coevolving red and blue simulation parameters. In *Proceedings of the 37th conference on Winter simulation*. Winter Simulation Conference, 1008–1012.

[16] Roger B Myerson. 2013. *Game theory*. Harvard university press.

[17] Marek Ostaszewski, Franciszek Seredynski, and Pascal Bouvry. 2007. Coevolutionary-based mechanisms for network anomaly detection. *Journal of Mathematical Modelling and Algorithms* 6, 3 (2007), 411–431.

[18] Una-May O'Reilly and Erik Hemberg. 2018. An Artificial Coevolutionary Framework for Adversarial AI.. In *AAAI Fall Symposia*.

[19] Marcos Pertierra Arrojo. 2018. Investigating coevolutionary algorithms For expensive fitness evaluations in cybersecurity. (2018).

[20] Elena Popovici, Anthony Bucci, R Paul Wiegand, and Edwin D De Jong. 2012. Coevolutionary principles. *Handbook of natural computing* (2012), 987–1033.

[21] Christopher D Rosin and Richard K Belew. 1997. New methods for competitive coevolution. *Evolutionary Computation* 5, 1 (1997), 1–29.

[22] George Rush, Daniel R Tauritz, and Alexander D Kent. 2015. Coevolutionary Agent-based Network Defense Lightweight Event System (CANDLES). In *Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference*. ACM, 859–866.

[23] Spyridon Samothrakis, Simon Lucas, ThomasPhilip Runarsson, and David Robles. 2013. Coevolving game-playing agents: Measuring performance and intransitivities. *IEEE Transactions on Evolutionary Computation* 17, 2 (2013), 213–226.

[24] Daniel Prado Sanchez, Marcos A Pertierra, Erik Hemberg, and Una-May O'Reilly. 2018. Competitive coevolutionary algorithm decision support. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 300–301.

[25] Sevil Sen, Emre Aydogan, and Ahmet I Aysan. 2018. Coevolution of Mobile Malware and Anti-Malware. *IEEE Transactions on Information Forensics and Security* 13, 10 (2018), 2563–2574.

[26] Karl Sims. 1994. Evolving 3D morphology and behavior by competition. *Artificial life* 1, 4 (1994), 353–372.

[27] You Seok Son and Ross Baldick. 2004. Hybrid coevolutionary programming for Nash equilibrium search in games with local optima. *IEEE Transactions on Evolutionary Computation* 8, 4 (2004), 305–315.

[28] Forhad Zaman, Saber M Elsayed, Tapabrata Ray, and Ruhul A Sarkerr. 2018. Evolutionary algorithms for finding nash equilibria in electricity markets. *IEEE Transactions on Evolutionary Computation* 22, 4 (2018), 536–549.