Investigating Coevolutionary Archive Based Genetic Algorithms on Cyber Defense Networks

Dennis Garcia
MIT, CSAIL
32 Vassar St
Cambridge, Massachusetts 02142
dagarcia@mit.edu

Anthony Erb Lugo
MIT, CSAIL
32 Vassar St
Cambridge, Massachusetts 02142
aerblugo@mit.edu

Erik Hemberg
MIT, CSAIL
32 Vassar St
Cambridge, Massachusetts 02142
hembergerik@csail.mit.edu

Una-May O'Reilly
MIT, CSAIL
32 Vassar St
Cambridge, Massachusetts 02142
unamay@csail.mit.edu

ABSTRACT

We introduce a new cybersecurity project named RIVALS. RIVALS will assist in developing network defense strategies through modeling adversarial network attack and defense dynamics in peer-to-peer networks via coevolutionary algorithms. In this contribution, we describe RIVALS' current suite of coevolutionary algorithms that use archiving to maintain progressive exploration and that support different solution concepts as fitness metrics. We compare and contrast their effectiveness by executing a standard coevolutionary benchmark (Compare-on-one) and RIVALS simulations on 3 different network topologies. Currently, we model denial of service (DOS) attack strategies by the attacker selecting one or more network servers to disable for some duration. Defenders can choose one of three different network routing protocols: shortest path, flooding and a peer-to-peer ring overlay to try to maintain their performance. Attack completion and resource cost minimization serve as attacker objectives. Mission completion and resource cost minimization are the reciprocal defender objectives. Our experiments show that existing algorithms either sacrifice execution speed or forgo the assurance of consistent results. rIPCA, our adaptation of a known coevolutionary algorithm named IPCA, is able to more consistently produce high quality results, albeit without IPCA's guarantees for results with monotonically increasing performance, without sacrificing speed.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability;

KEYWORDS

cybersecurity, coevolution, network, genetic algorithms, evolutionary algorithms

GECCO'17, July 2017, Berlin, Germany

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

ACM Reference format:
Dennis Garcia, Anthony Erb Lugo, Erik Hemberg, and Una-May O'Reilly. 2017. Investigating Coevolutionary Archive Based Genetic Algorithms on Cyber Defense Networks. In *Proceedings of ACM GECCO conference, Berlin, Germany, July 2017 (GECCO'17),* 8 pages.
https://doi.org/10.475/123_4

1 INTRODUCTION

Cyber attacks continue to increase in frequency and severity and have been the cause of numerous disruptions in both industry and politics. With more and more critical information moving through networks, it is important to make sure that defenses are in place to help keep these networks secure. When an attacker is deterred by a specific defense, the attacker usually changes strategies and is then able to wreak havoc once again. Defenders are then forced to adjust to these new attacks and an entire adversarial process iterates and escalates. In today's landscape, there are many networks that are set up without adequate capability to fend off such *adaptive* attacks.

We introduce a new cybersecurity project named RIVALS. Rather than manually tune and conjure up defenses for a network every time an attacker adapts and acts in a novel way, RIVALS is intended to use coevolutionary algorithms to determine the best defense for a network amidst constantly changing cyber attacks, see Figure 1. In particular, RIVALS will focus on how a peer-to-peer network can be deployed as a robust and resilient means of securing mission reliability in the face of extreme distributed denial of service attacks. RIVALS' premise is that leveraging the decentralized properties of peer-to-peer networks with enhanced capabilities will allow a mission to complete despite sustaining an attack.

RIVALS will eventually include a peer-to-peer network simulator that runs an extended version of the Chord [19] peer-to-peer protocol. In this contribution, because of the early point in the project's course, RIVALS operates with the original Chord protocol. We model simple attacks and defenses on a network. We measure the performance of attackers and defenders through the concept of a mission. A mission, for our purposes, represents a set of tasks to be completed. These tasks rely on the network's health for their success. An attacker's goal is to degrade the network such that the tasks, and ultimately the mission, fail. Meanwhile, a defender's goal

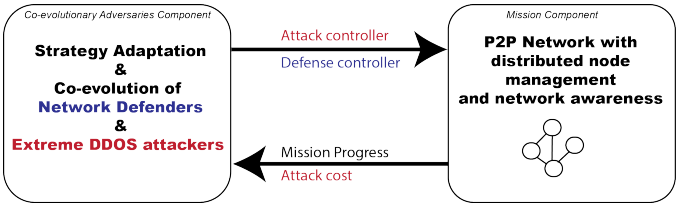

Figure 1: RIVALS system overview.

is to ensure the success of the mission. This setup is useful as it allows us to abstract and simulate a real-world adversarial environment. To model the co-adaptive behavior of adversaries, we regard our attacking and defending algorithms as populations under the direction of a coevolutionary algorithm such as IPCA[11]. Over the course of many generations, a co-evolutionary optimization process reveals strong defender and attacker strategies in juxtaposition as well as dynamics. This rich history can then be reviewed to determine an effective defensive protocol for a given network. Multiple runs of coevolutionary algorithms, started from different initial conditions, can provide further insights when their outcomes significantly differ.

In this contribution, we present (1) RIVALS' current coevolutionary algorithms, (2) the grammatical representation we intend to exploit to model attacker and defender behavior, and (3) Chord, the peer-to-peer network protocol RIVALS will eventually leverage for defensive strategies. One coevolutionary algorithm, rIPCA, is new. It is a modified version of IPCA in which we apply a non-dominated filter to both coevolving populations, learners (defenders) and tests (attackers), as opposed to just the learner population in IPCA. This filter, presented in IPCA, extracts the individuals of a population which are not dominated by other individuals within the population, thus selecting a pareto front of the filtered individuals.

Through experiments, we examine the performance of the different archive-based coevolutionary algorithms by using RIVALS' current network simulator. To obtain a baseline, we preliminarily compare them using the Compare-on-one problem. Our metrics for Compare-on-one are execution time and final performance. For network simulations, we determine the usefulness of rIPCA and the other coevolutionary algorithms by comparing how they each perform if only one of the attacker or defender population evolves and if both populations evolve with coevolution. This comparison is conducted on 3 different network topologies.

The rest of this paper is organized as follows. In section § 2, we introduce similar work as well as necessary background information on peer-to-peer attacks and coevolutionary algorithms. Next, in section § 3, we present a brief overview of our method. Section § 4 presents the results from our experimentation. Section § 5 concludes the paper and discusses potential future directions of the project.

2 RELATED WORK

RIVALS is an example of the study of adversarial dynamics. Multiple sub-fields within Artificial Intelligence study this topic with different approaches: (A) Evolutionary Computation uses its biologic metaphor to focus upon adaptation, e.g. botnet detection system analysis and the effect of botnet evolution [7], infrastructure resilience through competitive coevolution [18], and a critical infrastructure model analyzed via coevolutionary strategies [10]. (B) Machine Learning relies upon observational data and is often forensic rather than anticipatory, e.g. email spammers gaming against email spam-filters as in [4] (C) Game theory is used to find optimal strategies in an adversarial setting, e.g the study of autonomous, collaborative control for resilient cyber defense [20] which employs a distributed, game-theoretic approach to apportion computational loads in an efficient, prioritized, Pareto-optimal fashion among geographically dispersed clouds. (D) AI-Planning realizes autonomous agent strategies defined by goals, e.g. UAV path re-planning under critical situations [3].

Moving target defenses (MTD) in cybersecurity have drawn noteworthy attention because they aim to keep an adversary off guard by system state changes that do not allow attackers time to focus on a vulnerability or a complete multi-step plan. The challenge is to make the movement without excessively disrupting normal operations and interactions. RIVALS bears similarity to MTD systems because it adapts defenses, though it uses evolutionary computation as its adaptive mechanism rather than stochasticity, making it responsive to attack strategies. Evolutionary computation has been used in MTD research. In [22] for example, attackers (not the defense) are evolved with a genetic algorithm.

RIVALS aims to be original by occupying an underrepresented niche that combines coevolution and network modeling and simulation. It focuses on anticipatory insights as opposed to reactively mining a historical dataset once a type of attack has been discovered. It sets up a realistic model of cyber's asymmetric competitor action spaces. Specifically, attackers compete and adapt with one action repertoire and defenses use with another. We can think of only a few modest parallels of this model. In baseball, a player bats offensively and fields defensively using distinctly different competitive skills. Throughout the course of a game, as one side's offense is pit against the other's defense, there is an evolution and adaptation of strategies by both sides. The CANDLES system bears close resemblance to RIVALS [17]. CANDLES, Coevolutionary Agent-based Network Defense Lightweight Event System, is a framework designed to competitively coevolve *both* attacker and defender agent strategies. Like RIVALS, it employs coevolutionary genetic algorithms and its competitor action space is asymmetric. In contrast, it is a quite abstract *network security* simulation. Attacker solutions consists of a list of target machines, reconnaissance techniques, and exploits while defender solutions use a paranoia threshold, a budget, a list of suspected targets, detection systems and dynamic mitigations. Surprisingly, a survey of literature in coevolutionary computation (not confined to cybersecurity applications) only yields one such asymmetric setup: that of coevolving programs and unit tests from their specification, e.g. [1].

Finally RIVALS is the successor to *STEALTH (Simulating Tax Evasion And Law Through Heuristics)* [9]. STEALTH is a coevolutionary and modeling methodology for exploring how non-compliant tax strategies evolve in response to abstracted auditing and regulatory attempts that evolve to detect them.

The next section describes coevolutionary search in adversarial cybersecurity, coevolutionary algorithms based on archives, how to represent solutions with grammars, and the Chord protocol.

3 METHOD

In this section, we present RIVALS with respect to the following topics: coevolutionary algorithms, grammars, and the Chord protocol.

3.1 Coevolutionary Algorithms

Coevolutionary algorithms explore domains in which the quality of a *solution* is determined by its performance when interacting with some set of *tests*. Reciprocally, a *test*'s quality is determined by its performance when interacting with some set of *solutions*. For example, the tests of a network attack strategy are different network routing behaviors that could repel the attack, and reciprocally the tests of a network behavior are different attack strategies that could disrupt the network.

Because a solution's performance is measured over multiple tests, its fitness can be quantified in a number of ways and may change depending on what tests it faces. This complexity has been extensively addressed in coevolutionary literature, e.g. [2, 16].

RIVALS integrates 4 *solution concepts*, [16] in total, mixed across different coevolutionary algorithms: (1) **Best Worst Case** A *solution*'s fitness is its worst performance measure against the fittest *test* in the set of *tests* that it tries to solve (2) **Maximization of Expected Utility** A solution's fitness reflects that its tests are of equal importance. (3) **Nash Equilibrium** favors solutions which lead to stable solution states in which no sole actor can their improve their state unilaterally. (4) **Pareto Optimal Set** Every possible *test* (*solution*) is an objective and the subset of *solutions* (*tests*) are the pareto set of this multi-objective space. Pathologies arise in coevolutionary optimization due to its complex dynamics, [12]. These include: (1) Intransitivity, e.g. (a) Red Queen Effect (b) Cycling (c) Transitive dominance, and, (2) Disengagement (loss of gradient), e.g. (a) *solution* fails to perform in any way on a *test* (b) inability to discover a *test* to efficiently search for *solutions*.

One general remedy for these coevolutionary search pathologies is memory which can be served by the maintenance of an archive. An archive is a repository of solutions that is maintained outside the algorithmic cycle of generational selection and variation. It retains the best solutions ever found [5, 11, 12, 14] thus serving as a source of genetic material that persists longer than a generation and as a source of individuals with historical performance characteristics that are saved from being lost due to genetic variation. Coevolutionary algorithms have an archive for each population.

3.1.1 Coevolutionary Algorithms with Archives. RIVALS includes as baselines (1) a simple coevolutionary algorithm that does not use an archive and (2) an implementation of IPCA, Pareto-Coevolution Archive,[11]. IPCA archives previous tests and only replaces them with new tests which are different and more competitive than those in the archive. The learner (a.k.a. solution) archive is maintained by selecting learners that are not dominated by other learners in terms of which tests they solve. That is, if a learner, X, only solves tests A and B, and learner, Y, only solves test A, then learner X dominates Y and Y is removed from the learner archive. This strategy fosters monotonic evolutionary progress. We validate our implementation of IPCA and the other coevolutionary algorithms with the discretized three-dimensional Compare-on-one problem with mutation bias.

`Coev`, a.k.a Algorithm 1, is pseudocode of our simple coevolutionary algorithm [9]. It can be configured to use either the maximum expected utility solution concept or the best worst solution concept. IPCA and rIPCA, a.k.a. Algorithm 2, use archives and the Pareto Optimal Set solution concept. rIPCA applies the Pareto Optimal Set solution concept to both populations, as opposed to just the learner population as done in IPCA(see ALG.2 line 9). MaxSolve, a.k.a. Algorithm 3 from [5] uses the maximum expected utility solution concept and archives.

Algorithm 1 `Coev`

```
1:  procedure Coev(populations, generations)
2:      t ← 0
3:      best_individuals ← ∅
4:      while t < generations do            ▷ run for # generations
5:          pop′ ← Generate(populations)
6:          if BestWorstCase then
7:              pop′ ← EvalBestWorstCaseFitness(pop′)
8:          if MaximumExpectedUtility then
9:              pop′ ← EvaluateMEUFitness(pop′)
10:         populations ← Merge(populations, pop′)
11:         populations ← SortPopulations(populations)
12:         best_individuals ← ExtractBest(populations)
13:         t ← t + 1
14:     return best_individuals     ▷ Returns best solutions found
```

3.2 Grammatical representation for coevolutionary search

RIVALS will eventually exploit complex grammars to facilitate the expression and exploration of complex attack sequences and defender strategies. This will also help with incorporating domain knowledge. It uses Grammatical Evolution (GE) as its method. GE uses a variable length integer representation that maps from a grammar[15]. An example of GE and competitive coevolution is in the investigation of spatial coevolution of age layered planes in robocode[8]. The ease of use of GE currently outweighs our concern regarding the low locality of GE operators, e.g. [21].

The current RIVALS attack grammar is simple. Given start symbol `<Attacks>` , it is:

⟨*Attacks*⟩ `::= DOSAttack(`⟨*node*⟩`,` ⟨*start_time*⟩`,` ⟨*end_time*⟩`)`
`|` `DOSAttack(`⟨*node*⟩`,` ⟨*start_time*⟩`,` ⟨*end_time*⟩`),` ⟨*Attacks*⟩

⟨*node*⟩ `::= 0 | 1 | 2 | 3 | 4 | 5 | 6`

⟨*start_time*⟩ `::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`

⟨*end_time*⟩ `::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`

The nodes that an attacker can attack and the duration of the attacks vary between different topologies. The grammar is recursive allowing multiple nodes to be attacked without pre-specifying many. The representation for the defense is currently a simple choice of one of three functions each implementing a different routing protocol.

Algorithm 2 IPCA, rIPCA

```
1:  procedure IPCA(populations, generations)
2:      t ← 0
3:      L^0 ← populations_learners
4:      T^0 ← populations_tests
5:      best_individuals ← ∅
6:      while t < generations do                    ▷ run for # generations
7:          T^t ← NonDominated(L^t, T^t)            ▷ extract pareto-front
8:          if rIPCA then
9:              L^t ← NonDominated(T^t, L^t)        ▷ extract pareto-front
10:         L^{t+1} ← L_t
11:         T^{t+1} ← T_t
12:         NL ← GenerateLearners(L^t)
13:         NT ← GenerateTests(T^t)
14:         TS ← UsefulTests(NT, T^t, NL, L^t)
15:         T^{t+1} ← T^{t+1} ∪ TS
16:         for i = 1..|NL| do
17:             if Useful(L_i, L^{t+1}, T^{t+1}) then
18:                 L^{t+1} ← L^{t+1} ∪ L_i
19:         if L^{t+1} ≠ L^t then
20:             t ← t + 1
21:         best_individuals ← ExtractBest(populations)
22:     return best_individuals                     ▷ Returns best solutions found
```

Algorithm 3 MaxSolve

```
1:  procedure SUBMIT(LN, TN)
2:      L ← L ∪ LN
3:      T ← T ∪ TN
4:      n_solved ← {}
5:      for l ∈ L do
6:          n_solved[l] ← NumberSolved(l, T)
7:      for (l_i, l_j) ∈ L^2, i < j do
8:          if ∀t ∈ T : G(l_i, t) = G(l_j, t) then n_solved[l_j] ← 0
9:      Sort(LN, n_solved)
10:     for l ∈ L do
11:         if n_solved[l] > 0 then
12:             Select(l)
13:     for t ∈ T do
14:         if ∃l ∈ L : Solves(l, t) then
15:             Select(t)
16:     for t ∈ T do
17:         for t' ∈ T, t'! = t do
18:             if ∀l ∈ L : G(l, t) = G(l, t') then
19:                 Deselect(t)
20:     return L, T                                 ▷ Returns updated populations
```

3.3 Peer-to-Peer Network

As mentioned in § 1, RIVALS will focus on defensive strategies based upon a peer-to-peer network. A peer-to-peer network has no single point of failure and thus is inherently more robust to defend against DDoS attacks than other types of networks. While there are other peer-to-peer protocols that RIVALS could have used as overlays of a virtual network on a physical network, we chose Chord because it is efficient and extensible. Chord is also documented with pseudocode making it easier to implement a model of it that runs on our network simulator.

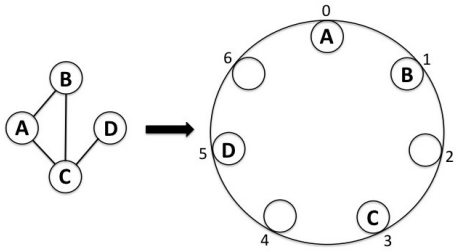

Figure 2: Physical network on the left and its virtual Chord overlay representation on the right.

We now briefly describe important elements of the Chord protocol (for more details see [19]) and our implementation of it. At its core, Chord relies upon distributed hashing via node-based finger tables. These tables hold information that helps logarithmically decrease the cost of finding which node holds a queried key. Each key is associated with some information. Chord assigns each node and key m-bit identifiers and places each identifier in an identifier circle. We illustrate this process in Figure 2: each node in a physical network is given an identifier and logically placed, i.e. becomes a virtual node, in a location in the identifier circle, i.e. overlay network. Each key in the identifier circle is assigned a node. The key's identifier is either equal to that of the node or the node that immediately follows the identifier of the key in the circle.

For example, in Figure 2, if a key were given the identifier of 2, the node responsible for this key would be the node at identifier 3. Each node maintains extra information in its finger table that it uses to direct key queries efficiently along the circle. A lookup sends the query at least halfway to its destination by taking advantage of finger table information. Chord handles nodes entering and dropping off the network by updating node finger tables periodically as well as reapportioning keys around the network as needed. This stabilization naturally lends itself to adversarial situations where an adversary intentionally forces nodes out of availability.

We currently simply model Chord on a single workstation. Upon nodes leaving or joining the network, the original Chord protocol *eventually* stabilizes itself through periodic actions. In contrast, in RIVALS' implementation every time a node leaves or joins the network, successor and predecessor pointers as well as the finger tables are immediately repaired. Another contrast is that nodes in the Chord network become part of the circle by receiving an m-bit identifier obtained by hashing the nodes with SHA-1. In RIVALS' implementation, Python's builtin random library is used instead.

4 EXPERIMENTS

Our experiments seek to understand the capabilities of our coevolutionary algorithms. They help us start to examine and interpret the dynamics that result from them in the context of network security.

Table 1: Algorithm Settings

Parameter Setting	Compare-on-one	Network Simulations
Population size	10	40 (10 for Topology 2)
Archive size	10	20
Generations	1000	20
Max length	10	20
Parent archive probability	0.9	0.9
Crossover probability	0.8	0.8
Mutation probability	0.1	0.1
Mutation bias low	-0.15	NA
Mutation bias high	0.1	NA
Generation loop breakout	500	NA
Grammar	No	Yes

4.1 Setup

We first benchmark the algorithm suite using the Compare-on-one problem. Next, we apply it in a simple RIVALS context by setting up 3 different network topologies run with network simulation.

Settings of the parameters for the algorithms used for the experiments are in Table 1. The settings for Compare-on-one are taken from [11] for standardization and comparative reference. The settings for the network simulations are chosen as an initial exploration. We present average results over 30 runs.

4.1.1 Compare-on-one. In the Compare-on-one problem [11], each population is made up of individuals that are represented as real valued n dimensional vectors. We compare one individual, *test A*, against an individual, *solution B*, of the opposing population by determining which dimension in B's vector has the highest value and if A's corresponding value (by dimension) is greater than or equal to B's. Performance of a population is then measured, at each generation, by taking the minimum of the minimum values of each individual's vector representation. Note that we use competitive fitness sharing as in the IPCA paper[11]. The mutation bias settings are included in Table 1.

4.1.2 Network Simulations. We give an overview of the context, terms and components of the network simulations including: the network topology, missions, attacker goals and capabilities, and defender goals and capabilities.

Context: DDoS attacks are a common way to disrupt certain network resources and are accomplished by flooding the target with a high volume of traffic. For example, like a SYN FLOOD[1] attack. The attack grammar, § 3.2, allows nodes to be selected and flooded.

Network Topology We start with a simple topology (Figure 3, Topology 0) as a benchmark that allows us to explore simple mission scenarios exhaustively before scaling up to larger and more realistic topologies (Figures 4 & 5, Topologies 1 and 2) that are too large to conveniently enumerate all the combinations of attacks.

Missions: A mission will eventually be comprised of a sequence of tasks where each task has a start node, an end node, and a maximum duration after which it fails. Tasks are meant to simulate different parts of a mission, the parts relevant to a network could be e.g. coordination via chat between two users, using Internet Relay Chat (IRC), or transfer of a file using File Transfer Protocol (FTP) from one user to a server. A mission is successful if every task is completed one after the other in the time allowed per task. It is unsuccessful if any of the tasks of the mission fail. Currently, missions are limited to one task to allow us to reason about the results obtained.

[1] https://en.wikipedia.org/wiki/SYN_flood

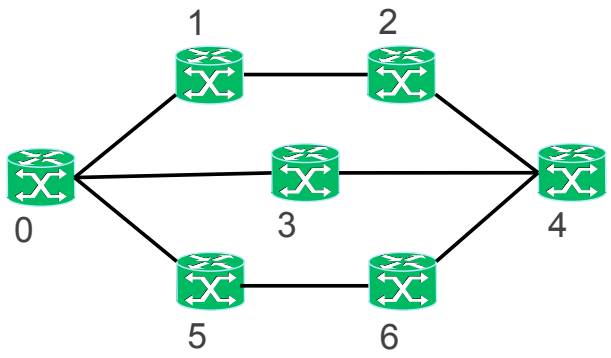

Figure 3: Topology 0, simple network, used to benchmark the defensive actions for routing of Shortest-path, Flooding and Chord

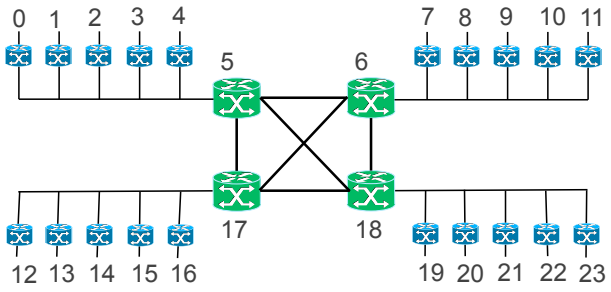

Figure 4: Topology 1, larger network providing more nodes and a different topology

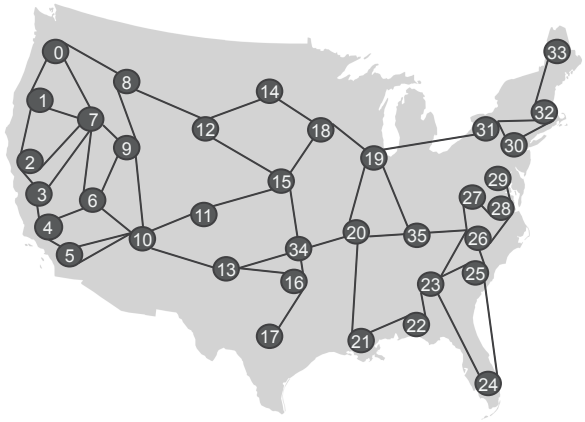

Figure 5: Topology 2, possible network for a more realistic mission

Attacker: The goal of the attacker is to disrupt the network, with as little effort as possible, directing its DOS attacks in a way that causes mission failure. The attacker is quite powerful and can do this through its capability of being able to specify any node or set of nodes in the network to launch DDoS attacks on. The attacker has the ability to cause failure in these nodes and specify for how long it wants the DoS attack on each node to last. (See grammar in § 3.2)

Defender: The goal of the defender is to ensure mission success. The defender currently does this by choosing among 3 different routing protocols that use different techniques to deal with the nodes being out of service (attacked): **Shortest path protocol** At the beginning of a task, the network calculates the shortest path from a start node to an end node, and attempts to send the packet along this path. If at any point along the way the path becomes blocked due to node failure caused by an attacker, the network waits for the blocked node to become free before continuing. This protocol is more expensive in terms of time when a network is under attack. It is also more vulnerable to single nodes being attacked. **Flooding protocol** The flooding protocol works by sending multiple copies of the packet along all available paths and completes the task when the first packet reaches its destination through any of these paths. This is more expensive in hops but could be cheaper in time when under an attack.

Chord protocol Chord chooses paths using its finger tables. Even under attack, its routing persists due to its reconfigurability when a node is lost or returns to service (see § 3.3).

Fitness Functions: We defined fitness functions that reflect the goals of the attacker and defender. We reward attackers for being able to disrupt a mission by attacking very few nodes for a short amount of time and punish attackers as the number of nodes and for how long they attack them increases. The fitness function for the attacker is

$$f_a = \frac{1 - mission\_success}{(n\_attacks \cdot total\_duration) + n\_attacks}$$

where $mission\_success$ is describing whether the entire mission succeeded(1) or failed(0), $n\_attacks$ is the total number of nodes attacked in the network, and $total\_duration$ is the aggregated amount of time nodes were attacked. We include an additional $n\_attacks$ term in the denominator so as to prefer solutions with least amount of attacks. Note that with the grammar used (Section § 3.2), the fitness function rewards attacker that attack fewer nodes, even though the recursive grammar allows any number of them.

Similarly, we reward defenders that complete the mission quickly and with a short amount of hops and punish those that take longer and use more network resources. For example, the flooding routing mechanism gives a better guarantee that the mission will be completed than the shortest path protocol, but floods the network and thus uses many hops around the network to do so. This behavior is taken into account into the fitness function and punished.

Table 2: Results regarding execution time and performance for Compare-on-one for the different coevolutionary algorithms.

Algorithm	Exec Time(s)	Final Perf.
Coev	7.115 ± 0.219	0.538 ± 0.297
MinMax	7.370 ± 1.962	0.481 ± 0.281
MaxSolve	13.901 ± 1.044	0.530 ± 0.295
IPCA	335.742 ± 114.220	0.870 ± 0.225
rIPCA	54.479 ± 47.474	0.806 ± 0.257

The fitness function for the defender is

$$f_d = \frac{mission\_success}{overall\_time \cdot n\_hops}$$

where $overall\_time$ is the total time a specific routing protocol took to complete the mission and n_hops is total number of hops taken by the protocol to complete the mission.

In order to keep the network simulation simple, we assume that every edge is unit-length.

4.2 Results

In terms of the performance of the different algorithms on the Compare-on-one problem, we expected the rIPCA algorithm to be the most promising as rIPCA builds upon the algorithms mentioned in this paper and also aims to eliminate redundancies within archive populations thus decreasing overall runtime. In terms of running the algorithms on our simulated scenario, we expected to see the algorithms name the Chord protocol implementation as the network defense mechanism that is best able to handle network attacks.

4.2.1 Compare-on-one. Table 2 show results for Compare-on-one on different algorithms. The IPCA and rIPCA algorithms have the better performance. This is expected as they both maintain archives with monotonically increasing fitness throughout their execution. We also note that rIPCA's execution time is notably shorter than the execution time for IPCA. It follows from the implementation of rIPCA as rIPCA implements IPCA but also applies the non-dominance filter on its test individuals. This works to limit the size of the test population. With the size of the test population limited, rIPCA is able to run a modified IPCA algorithm with similar performance but many less operations. The execution times show that IPCA and rIPCA take the longest, but IPCA is significantly slower.

4.2.2 Network Missions. Prior to running the experiments regarding network missions, we established a baseline by an exhaustive search of Topology 0 (Figure 3). We set attacks to last the full duration of a task and saw that Chord only fails if all the nodes are attacked, shortest path fails if any node on the shortest path is blocked, and flooding is blocked if start, end or when all paths contain a node that is blocked. We performed this exhaustive search so we could later verify the correctness of both the algorithms and the defense protocols in the network and to show that this is possible in topologies with a small number of nodes but becomes increasingly difficult for a topology as large as the ones in Figures 4, 5.

The columns of Table 3 represent the following. The attack columns represent the results from running each of the coevolutionary algorithms on a fixed defense protocol (in this case the Chord protocol in each case). Similarly, for the defense columns, we instead fix the attack on the network and apply the coevolutionary algorithms as we did with the attack columns. Finally, we run the coevolution between populations of attackers and defenders on each algorithm and report these results in the coevolution columns. We perform these experiments across the three topologies presented earlier. The entries of the table represent the state of each population at the end of a run under one of the implemented coevolutionary algorithms by calculating the average fitness of the defending population. We run each algorithm and topology setting 30 times and report averages over the runs.

When fixing the defender as the Chord protocol across all algorithms on Topology 0, we see that our simple coevolutionary algorithm, `COEV`, is less able to deter the mission. This explains the high average fitness value for the defender population in the first entry in the attack column. The decreasing trend of defender average fitness values down this column shows that algorithms such as IPCA and rIPCA seem to perform better at finding strong attack individuals. In the defense column of Topology 0, in the case of a fixed attack, we notice, since the Chord protocol evaluates to 0.250 fitness against the fixed attack, that most algorithms converge on the Chord protocol as the best defense. In the coevolution column, we notice how the algorithms can have a large effect when both populations are set to maintain archives. We conjecture this is due to the nature of the test archives for both IPCA and rIPCA as these archives help enforce monotonic performance increases.

In the columns of Topology 1, we first note how the average fitness values of the defender populations are lower across the entire attack column than they were under Topology 0. We reason that this change happened not because of the performance of the algorithms rather the increase in the number of nodes in the topology. This follows since the defenses' fitness function is inversely proportional to the number of hops needed to reach its goal. Thus, an increase in the size of the topology will lead to an overall decrease in fitness values for the defense population. The defense column for Topology 1 is similar to that of the defense column for Topology 0 as they all converge to the Chord protocol. As for the coevolutionary results, we note that both IPCA and rIPCA converged on a solution as evidenced by their low standard deviations. The other three algorithms show moderate performance, but produce results with more variance.

We now consider Topology 2 – the largest of the topologies. In the attack column, we notice the same decreasing trend we saw before in the attack column of Topology 0. This helps reinforce our claim that IPCA and rIPCA produce better results in general. While the results in the defense column closely resembled those of the defense columns in the previous topologies, we did notice that MaxSolve was unable to converge on a solution. Furthermore, the coevolution results are much like the ones of Topology 1 in that both IPCA and rIPCA converge on a result.

In Figure 6, we further explore the coevolution process by examining the average fitness values for both attack and defense populations from one of our sample runs over Topology 0 while

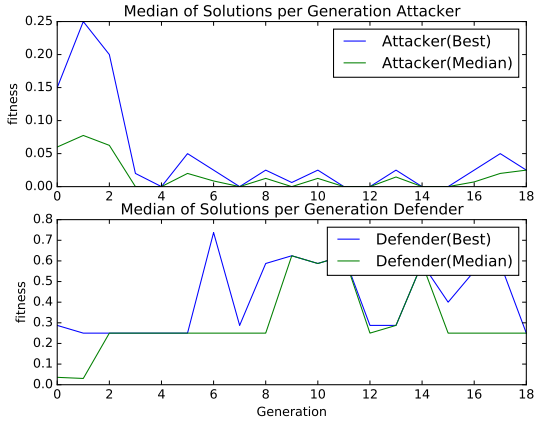

Figure 6: Results from sample run on network topology 0 under the simple COEV algorithm. Top: Median and best fitness results for attacker population over 20 generations. Bottom: Median and best fitness results for attacker population over 20 generations.

running the simple coevolution algorithm. In the attacker's average fitness graph, we note that the average fitness for the attack population experiences a short boom in performance followed by a quick drop to 0.00. It then proceeds to oscillate as the defense population converges on the chord protocol. The oscillation behavior follows from how the defense population evolves throughout the run. The variation in the population introduced by the evolution after generation causes non-optimal (i.e. non-chord) solutions to form part of the defense population. This, in turn, increases the average fitness for attacker individuals as they face defenses which they can succeed against. Additionally, the attacks introduced may force the chord protocol to explore paths which are shorter than its original path, thus increasing the performance of the chord protocol occasionally.

Overall, the results for both the Compare-on-one experiments as well as the network mission experiments show that IPCA and rIPCA perform better in either scenario but are better suited at handling tasks where execution time isn't as important. We also show through our implementation of these coevolutionary algorithms that it is possible to model adversarial behavior on a network simulator. As expected, coevolution does not give as strong defenders as for a fixed attack.

5 CONCLUSIONS & FUTURE WORK

We have made progress in creating an end-to-end system where we have shown the ability to test the effectiveness of the different coevolutionary algorithms on simulated networks. Additionally, we plan to continue this work and have ambitious goals laid out for future versions of RIVALS. In particular, we are interested in defending against low intensity DDoS attacks[13]. Attacks like these are hard to detect because they can be sent in small waves and thus are not easy to spot amongst regular traffic patterns. One of our next tasks is to improve the realism of the network simulator by

Table 3: Network Mission results for attack, defense and coevolution for all topologies

	Topology 0			*Topology 1*			*Topology 2*		
Algorithm	**Attack**	**Defense**	**Coevolution**	**Attack**	**Defense**	**Coevolution**	**Attack**	**Defense**	**Coevolution**
Coev	1.000 ± 0.000	0.250 ± 0.000	0.227 ± 0.05	0.067 ± 0.000	0.100 ± 0.033	0.067 ± 0.031	0.173 ± 0.001	0.053 ± 0.022	0.053 ± 0.022
MinMax	0.975 ± 0.000	0.250 ± 0.000	0.200 ± 0.060	0.062 ± 0.000	0.100 ± 0.033	0.059 ± 0.013	0.102 ± 0.001	0.053 ± 0.022	0.057 ± 0.017
MaxSolve	0.826 ± 0.014	0.207 ± 0.088	0.263 ± 0.159	0.163 ± 0.002	0.111 ± 0.000	0.074 ± 0.026	0.096 ± 0.000	N/A	0.081 ± 0.027
IPCA	0.690 ± 0.079	0.250 ± 0.000	0.333 ± 0.063	0.073 ± 0.001	0.111 ± 0.000	0.070 ± 0.003	0.072 ± 0.000	0.062 ± 0.0	0.079 ± 0.000
rIPCA	0.698 ± 0.082	0.250 ± 0.000	0.463 ± 0.018	0.079 ± 0.008	0.111 ± 1.387	0.068 ± 0.003	0.073 ± 0.001	0.062 ± 0.000	0.062 ± 0.000

incrementally increasing its sophistication and complexity. We plan to do this by extending the Chord protocol, bridging the gap between the logical and physical representations of a network with a Chord overlay structure, testing the effect of more topologies on the performance of the defending network, and simulating attack types on both the logical and physical representation of the network. In future iterations of the simulator, we plan on writing a grammar to represent tasks in order to randomly generate missions of different number of tasks. Furthermore, we will continue to improve the performance and speed of the coevolutionary algorithms as well as try algorithms with all different solution concepts, e.g. Nash Equilibrium [6] and how to report fitness for different solution concepts and archives.

REFERENCES

[1] Andrea Arcuri and Xin Yao. 2007. Coevolving programs and unit tests from their specification. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. ACM, 397–400.

[2] Josh C Bongard and Hod Lipson. 2005. Nonlinear system identification using coevolution of models and tests. *IEEE Transactions on Evolutionary Computation* 9, 4 (2005), 361–384.

[3] Jesimar da Silva Arantes, Márcio da Silva Arantes, Claudio Fabiano Motta Toledo, and Brian Charles Williams. 2015. A Multi-population Genetic Algorithm for UAV Path Re-planning under Critical Situation. In *27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2015, Vietri sul Mare, Italy, November 9-11, 2015*. 486–493. DOI: http://dx.doi.org/10.1109/ICTAI.2015.78

[4] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, and others. 2004. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 99–108.

[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] Sevan G Ficici and Jordan B Pollack. 2003. A game-theoretic memory mechanism for coevolution. In *Genetic and Evolutionary Computation Conference*. Springer, 286–297.

[7] Fariba Haddadi and A Nur Zincir-Heywood. 2015. Botnet Detection System Analysis on the Effect of Botnet Evolution and Feature Representation. In *Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference*. ACM, 893–900.

[8] Robin Harper. 2014. Evolving robocode tanks for Evo robocode. *Genetic Programming and Evolvable Machines* 15, 4 (2014), 403–431.

[9] Erik Hemberg, Jacob Rosen, Geoff Warner, Sanith Wijesinghe, and Una-May OâĂŹReilly. 2016. Detecting tax evasion: a co-evolutionary approach. *Artificial Intelligence and Law* 24, 2 (2016), 149–182.

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

[11] Edwin D de Jong. 2007. A Monotonic Archive for Pareto-Coevolution. *Evolutionary Computation* 15, 1 (2007), 61–93.

[12] Krzysztof Krawiec and Malcolm Heywood. 2016. Solving Complex Problems with Coevolutionary Algorithms. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*. ACM, 687–713.

[13] Aleksandar Kuzmanovic and Edward W Knightly. 2003. Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, 75–86.

[14] Paweł 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] Michael O'Neill and Conor Ryan. 2003. *Grammatical evolution: evolutionary automatic programming in an arbitrary language*. Vol. 4. Springer.

[16] Elena Popovici, Anthony Bucci, R Paul Wiegand, and Edwin D De Jong. 2012. Coevolutionary principles. In *Handbook of Natural Computing*. Springer, 987–1033.

[17] 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.

[18] Travis Service and Daniel Tauritz. 2009. Increasing infrastructure resilience through competitive coevolution. *New Mathematics and Natural Computation* 5, 02 (2009), 441–457.

[19] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review* 31, 4 (2001), 149–160.

[20] Stuart Wagner, Eric Van Den Berg, Jim Giacopelli, Andrei Ghetie, Jim Burns, Miriam Tauil, Soumya Sen, Michael Wang, Mung Chiang, Tian Lan, and others. 2012. Autonomous, collaborative control for resilient cyber defense (ACCORD). In *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2012 IEEE Sixth International Conference on*. IEEE, 39–46.

[21] Peter A Whigham, Grant Dick, James Maclaurin, and Caitlin A Owen. 2015. Examining the Best of Both Worlds of Grammatical Evolution. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*. ACM, 1111–1118.

[22] Michael L Winterrose and Kevin M Carter. 2014. Strategic evolution of adversaries against temporal platform diversity active cyber defenses. In *Proceedings of the 2014 Symposium on Agent Directed Simulation*. Society for Computer Simulation International, 9.