Pre-Publication Version

Grammatical Evolution with Coevolutionary Algorithms in Cyber Security

Erik Hemberg Anthony Erb Lugo Dennis Garcia
Una-May O'Reilly

August 10, 2018

We apply Grammatical Evolution(GE), and multi population competitive coevolutionary algorithms to the domain of cybersecurity. Our interest (and concern) is the evolution of network denial of service attacks. In these cases, when attackers are deterred by a specific defense, they evolve their strategies until variations find success. Defenders are then forced to counter the new variations and an arms race ensues. We use GE and grammars to conveniently express and explore the behavior of network defenses and denial of service attacks under different mission and network scenarios. We use coevolution to model competition between attacks and defenses and the larger scale arms race. This allows us to study the dynamics and the solutions of the competing adversaries.

1 Introduction

Cyber attacks have increased in frequency, sophistication, and severity, and have been the cause of numerous disruptions. Denial of service attacks target computer networks because critical data and transactions now flow through them. As a result, it is crucial to not only be aware of the capabilities of cyber attackers, but also to design more secure networks. The issue with the current state of cyber defenses, however, is that they are largely reactive in nature. It is sometimes only when an attack is experienced that a network defense is strengthened. When attackers consequently alter their strategies, the of reactive defensive behavior repeats. Our goal is to investigate this coevolutionary arms race in order to shed light on its dynamics and identify robust defenses in advance of deployment.

Grammatical Evolution, see Figure 1, is initialized with a grammar expressed in Backus Naur Form (BNF) and search parameters. The grammar describes a language in the problem domain and its (rewrite) rules express how a sentence, i.e. solution, can be composed by rewriting a start symbol, i.e. high level goal. In our system's GE component, the BNF description, upon input, is parsed to a context free grammar representation. GE genotypes are fixed length integer vectors. Sentences of the grammar are GE phenotypes. To decode a genotype,

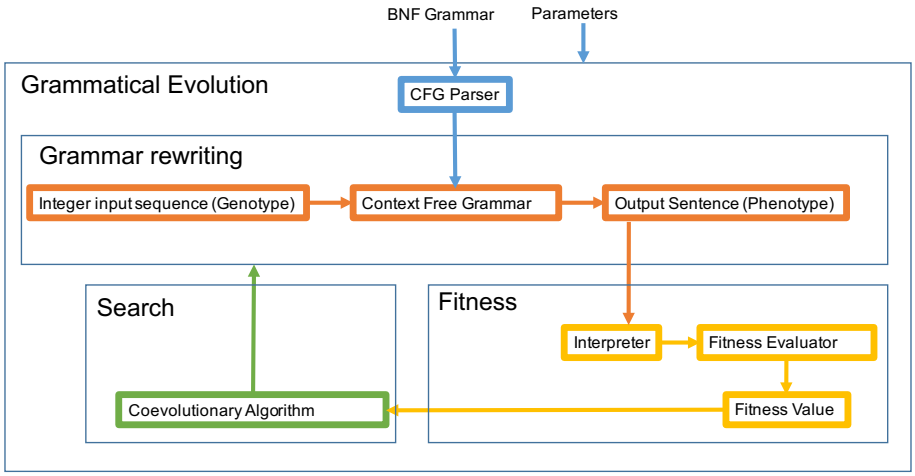


Fig. 1: Grammatical Evolution takes a BNF grammar and search parameters as input. The grammar rewrites the integer input to a sentence. Fitness is calculated by interpreting the sentence and then evaluate it. The search component modifies the solutions using two central mechanisms: fitness based selection and random variation. We use coevolutionary algorithms.

in sequence each of its integers is used to control the rewriting. This sentence is the phenotype. Fitness is calculated by interpreting the sentence and then evaluating it according to some objective(s). When we use our system to solve different problems, we only have to change the BNF grammar, the interpreter and the fitness function for each problem, rather than change the genotype representation. This modularity of GE and the reusability of the GE parser and rewriter are efficient software engineering and problem solving advantages. The grammar further helps us communicate our system's functionality to stakeholders because it enables conversations and validation at the domain level rather than at the algorithm level. This contributes to stakeholder confidence in solutions and our system.

Our system is named RIVALS [9]. Rather than manually tune and invent defenses for a network every time an attacker adapts and acts in a novel way, RIVALS assists during network design and hardening with the goal of anticipating attack evolution and identifying a robust defense that can circumvent the arms race and the reactive counter-measure postures. It uses coevolutionary algorithms [19] (and GE) to generate evolving network attacks and to evolve network defenses that effectively counter them, see Figure 2. RIVALS's research is grounded by focusing on peer-to-peer networks, specifically the Chord protocol, and extreme distributed denial of service (DDOS) attacks. A peer-to-peer network is a robust and resilient means of securing mission reliability in the face of extreme DDOS attacks. RIVALS' premise is that its attention to the coevolutionary nature of attack and defense dynamics will allow it to help iden-

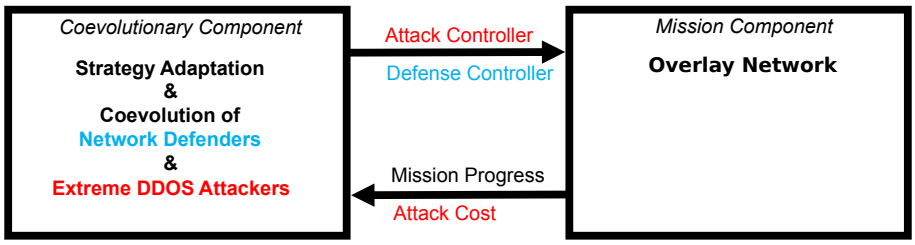


Fig. 2: RIVALS system overview. Defenders are marked in blue and attackers in red.

tify robust network design and deployment configurations that support mission completion despite an ongoing attack.

RIVALS currently includes a peer-to-peer network simulator that runs the Chord [22] protocol. It models simple attacks and defenses on networks running on the simulator. It measures the performance of attackers and defenders through the concept of a mission. A mission represents a set of tasks to be completed. These tasks rely on the network's quality of service to succeed. An attacker's goal is to degrade the network so that the tasks, and in extension the mission, fail. Meanwhile, a defender's goal is to ensure the success of the mission. Mission completion and resource cost metrics serve as attacker and defender measures of success. DDOS attacks in RIVALS are modeled as multiple nodes being selected and, from a start time, being completely disabled for some duration.

To model the co-adaptive behavior of adversaries, RIVALS sets up separate populations of attackers and defenders and coevolves them under the direction of a coevolutionary algorithm. Over the course of many generations, a coevolutionary optimization process reveals dual collections of more effective defender and attacker strategies. At this point in time RIVALS has a a suite of different coevolutionary algorithms with grammars for two simple problems. The algorithms explore archiving as a means of maintaining progressive exploration and support the evaluation of different solution concepts. All algorithms in our suite reuse the parser and rewriter component of GE.

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 RIVALS. Section 4 presents the our experimentation and Section 5 shows the results. Finally, Section 6 concludes the paper and discusses potential future directions.

2 Related Work

Our project investigates proactive cybersecurity modeling by means of GE and coevolutionary search algorithms. This section of related work and background

information considers cyber security, coevolutionary algorithms and GE in different combinations. In Section 2.1 we discuss projects at the intersection of evolutionary algorithms and cyber security, comparing them to RIVALS where relevant. In Section 2.2 we differentiate coevolutionary search algorithms from other EAs, independent of GE, and in Section 2.3 we discuss systems at the intersection of GE and coevolution. To date RIVALS is the only system that combines GE and coevolution in order to investigate a problem in the domain of cybersecurity.

2.1 Cyber Security and Evolutionary Algorithms

Moving Target Defense (MTD) projects, like RIVALS, use Evolutionary Algorithms (EAs) in a cybersecurity problem domain. The strategy of a MTD is to keep an attacker off guard by continually changing system configurations or information that the attacker needs to effectively attack. Strategies could involve changing the software underlying platforms, the location of sensitive data or the timing of system functions. For example, the system of [25] uses a GA to evolve adaptable adversarial strategies for defense against zero-day exploits. The system only adapts a defender population while RIVALS adapts both defender and attacker populations. In another contrast, it encodes strategies as binary chromosomes that represent finite state machines whereas RIVALS' uses a context free grammar. Arguably the important difference between RIVALS and this system is that evolution is used to address only two fixed scenarios while in RIVALS attackers compete with multiple defenders and defenders compete with multiple attackers.

Another work in this context and related to RIVALS is the coevolutionary agent-based network defense lightweight event system (CANDLES) [20]. It is designed to coevolve attacker and defender strategies in the context of a custom, abstract computer network defense simulation. CANDLES' attack and defense strategies are not expressed with grammars.

2.2 Coevolutionary Search

Coevolutionary algorithms are well suited to domains that have no intrinsic objective measure, also called *interactive* domains [19]. They can be distinguished as two types: *a)* Compositional coevolutionary algorithms that are used to solve problems where a solution involves interaction among many components that together might be thought of as a team. This is often called cooperative coevolution. *b)* Test-based coevolutionary algorithms that are used when the quality of a potential *solution* to the problem is determined by its performance when interacting with some set of *tests*. This is often called competitive coevolution.

Competitive coevolutionary algorithms are often applied in game search [19]. They are also related to game theory [19]. One advantage over game theory is that coevolutionary algorithms can be applied to larger search spaces [20].

There are a variety of examples of coevolutionary projects. One more theoretical project has investigated solution concepts for testcase coevolution with

a no free lunch framework [23]. Others address application domains such as streaming data classification [12], complexification of solutions [21], simulations of behavior and bug fixes [16].

2.2.1 Solution Concepts for Coevolutionary Algorithms

Coevolutionary algorithms differ from other EAs in one respect because they have two interacting populations. These dual populations imply that the algorithm explores domains in which the quality of a *solution* is determined by its performance when interacting with a set of *tests*. In return, a *test*'s quality is determined by its performance when interacting with some set of *solution*s. For example, the *tests* of a network attack strategy are different network routing behaviors that could resist the attack, and inversely the *tests* of a network behavior are different attack strategies that could disrupt the network.

Because a *solution*'s performance is measured against multiple *tests*, coevolutionary algorithms use *solution concepts* to express fitness and clarify what constitutes a superior solution [19]. Solution concepts include:

Best Worst Case A *solution*'s fitness is its worst performance against the set of *tests* that it tries to solve or its performance against the fittest test case. The coevolutionary algorithm's goal is to optimize the best worse case solution.

Maximization of Expected Utility A *solution*'s fitness is its average performance against the test cases. It is usually assumed that *tests* have equal importance. The coevolutionary algorithm's goal is to optimize the average case solution.

Nash Equilibrium Solutions which lead to stable *solution* states in which no sole actor can their improve their state unilaterally are preferred. The coevolutionary algorithm's goal is to find solutions at a Nash equilibrium.

Pareto Optimality Each *test* is considered to be an independent objective and a *solution* is a multi-dimensional datum in this multi-objective space. From this space, a pareto optimal (non-dominated) set of *solutions* can be identified as superior *solutions*.

It should also be noted that the interactive aspect of *solution* fitness also implies the algorithm lacks an *exact* fitness measurement. That is, usually, Evolutionary Algorithms rely upon a fitness function, a function of the form $f : G \mapsto \mathbb{R}$ that assigns a real value to each possible genotype in G. Individual solutions are compared as $f(g_0)$ with $f(g_1)$ and their relative ranking based on fitness is always the same, i.e. exact. In contrast, in coevolutionary algorithms two individuals are compared based on their interactions with other individuals. and because these individuals are only samples from a population and may change as a population undergoes evolution, the ranking of an individual relative to another solution is essentially an estimate.

We now describe a set of coevolutionary algorithm challenges and how they are remedied.

2.2.2 Coevolutionary algorithm challenges and remedies

Coevolutionary algorithms are challenging to work with because we have limited understanding of their detailed dynamics. Their two populations and dynamic solution concepts make them harder to interpret. [7]. The search driver, i.e. the selection pressure, is difficult to control because fitness measurements are only estimates. Fitness estimation makes it hard to precisely determine whether the algorithm is making productive progress. [19]. The problem of local optima still exists as it does with other EAs. The arms race we use coevolutionary algorithms for, in fact does not automatically appear since *tests* can be uninteresting, or not conform to some *a priori* goal [7]. This requires vigilant design and monitoring.

Coevolutionary search and optimization exhibits some unique pathologies that again arise from fitness being measured as a result of one or more interactions. One pathology is *intransitivity*, i.e. non transitive relations can exist between the competing solution spaces. [7]. For example, consider the intransitive cycle in Rock-Paper-Scissors where Rock beats Scissors, Scissors beats Paper, and Paper beats Rock. [14, 4, 7]. Some intransitive pathologies are:

Red Queen Effect Two populations continuously adapt to each other and their subjective fitness improves, but they fail to make any consistent progress along the objective metric. Conversely, they do make progress but the fitness estimate does not reflect this and falsely indicates a lack of progress. [4]

Cycling The adversary (whether *solution* or *test*) drops some element of selection pressure so abilities can be "forgotten" only to reuse them.

Transitive dominance One *solution* can be superior to a *test* that at the same time is superior to the *solution* according to a different conflicting subjective metric [4].

A general remedy to intransitivity is to maintain diversity and make sure an informative search gradient is always available. Another remedy is to explicitly assure that useful *tests* persist. This can be accomplished by introducing memory. Memory is usually implemented by means of an archive (see Figure 3), a repository of solutions that is maintained outside the algorithmic cycle of generational selection and variation, something like a *Hall of Fame*.

Another pathology is *disengagement* [4]. This occurs when one population is constantly superior to the other. At this point the subjective fitnesses of both populations become constant so there is no differential selection pressure and the search gradient is lost. Drift results.. Memory also helps address disengagement. Another remedy is to search explicitly for lower difficulty *tests* by looking for those which create less disagreement among *solutions* [4].

Memory, of course, also addresses cycling by preventing a *test* that selects for a *solution* from evolving out of the population.

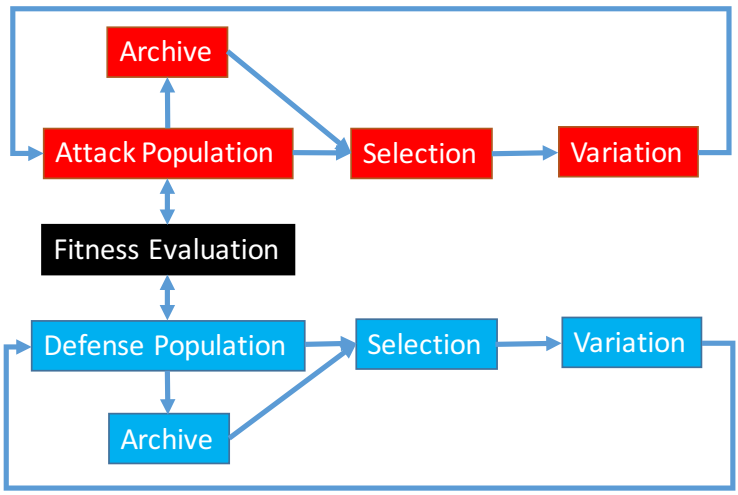


Fig. 3: A coevolutionary algorithm with two archives.

2.3 Coevolution and Grammatical Evolution

GE has previously been used in tandem with coevolutionary algorithms. In one use case, [6], coevolution and GE are used together to cope with dynamic environments. The GEGE method aims to find "modules" that can be reused when the environment changes, it has a compact representation of a larger grammar with an increased search space and strongly coupled grammars [3]. It uses cooperative coevolution to simultaneously evolve the grammar and genetic code with a hierarchy of grammars [6].

One study, [17, 18], used both GE and a Pareto-coevolutionary algorithm in a supervised machine learning context to train classifiers with a multi-objective fitness measure. It reformulated training data as a tw -population competition. Another study used both GE and coevolution to develop an Artificial Life model for evolving a predator–prey ecosystem of mathematical expressions [2]. Coevolutionary algorithms with GE have also been applied to financial trading using multiple cooperative populations [1, 8].

In one competitive coevolution and GE example, spatial coevolution in age-layered planes evolves robocode for robots [10]. In another, in the *STEALTH (Simulating Tax Evasion And Law Through Heuristics)* [11] project, a coevolutionary and modeling methodology is used to explore how non-compliant tax strategies evolve in response to abstracted auditing and regulatory attempts that evolve to detect them. STEALTH shares the arms race element of this chapter's work because in taxation, similar to cybersecurity, as soon as an evasion scheme is detected and stopped, a new, slightly mutated, variant of the scheme appears.

3 Methods

In this section we describe the methods we use. We present them in the following order: peer-to-peer networks in Section 3.1, the RIVALS network simulator in Section 3.2, coevolutionary algorithms that use archives in Section 3.3 and grammar representations for cybersecurity in Section 3.4. Because grammars are expressions of problem solving behavior, Section 3.3 also introduces the two problems we use to demonstrate RIVALS.

3.1 Peer-to-Peer Network

DDOS attacks often target a specific server within a network. By overloading this server with work, the server effectively becomes useless and the network struggles to route traffic through it. In a centralized network an attacker could attack its key central server, e.g. the server that is responsible for directing traffic to all the other servers, and take down the entire network. Peer-to-peer networks are not so fragile. They distribute data and resources with redundancy and thus have no single point of failure making them inherently more robust to DDOS attacks. Peer-to-peer networks are also robust to topological changes. They can continue to function even as nodes drop out as what may happen during a DDOS attack. They also can integrate additional nodes should they come back online. As an example, the Chord protocol includes a stabilization service handles nodes that are joining and leaving the network.

3.1.1 Chord Overview

We now briefly describe important elements of the Chord protocol (for more details see [22]) and our implementation of it. A peer-to-peer network is an overlay of a physical network. In Chord the logical network topology is a ring. Location-wise, each peer has a successor node and a predecessor node in the ring. Requests for data from a node need to be looked up to identify which node has the data and the node needs to be efficiently accessed. For lookup Chord uses distributed key hashing. For routing Chord relies upon node-based finger tables. A finger table is a look-up table for neighboring peers. Each table holds information that helps to logarithmically decrease the cost of finding which node holds a queried key. The Chord protocol includes a stabilization service handles nodes that are joining and leaving the network.

3.2 RIVALS Network Simulator

In RIVALS we currently model Chord on a single workstation. Upon nodes leaving or joining the network, the original Chord protocol *eventually* stabilizes itself through periodic actions. 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. In RIVALS nodes in the Chord network

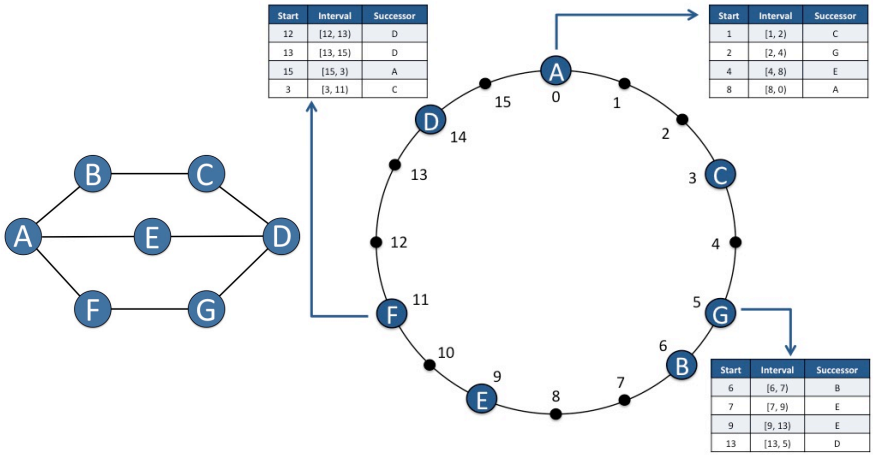


Fig. 4: Physical network on the left and its virtual(logical) Chord overlay representation on the right. The finger tables for nodes A, G, and F are shown.

become part of the circle by receiving an m-bit identifier obtained by hashing the nodes with SHA-1[1].

The network simulator has two versions. One version, we call the "logical" version, and the other the "logical to physical" version. The differences between these two versions of the software are explained in the next two subsections.

3.2.1 Logical network

The reason we name this version the "logical" version is because we assume the message gets sent via the hop series defined by the logical network. Figure 4 shows a simple physical network on the left, and the virtual (logical) Chord overlay network that gets constructed by the protocol on the right. In this example, if we are interested in finding a specific key in the network, we can ask any peer and that peer will use its finger table information to route the query to some peer that is closer to the target peer containing the desired key in the identifier circle. This series of queries provides a hopping series of the nodes visited before reaching the target node with the key. For example, if we ask node F where the key with identifier 3 is, it would pass the query to node A, and then node A would find that the key is located at node C. This results in a hopping series of F, A, C. In the simulator implementation, rather than use the protocol as a means of finding a key in the network, we use it as a means to represent the sending of a message through the network. We achieve this by asking the peer we consider the starting node, or the node responsible for sending the message, to find the identifier associated with the target node. In this sense, the difference is that we now use the protocol to lookup target node identifiers instead of key identifiers.

[1]In RIVALS' implementation, `Python`'s `random` library is used.

Tab. 1: Coevolutionary algorithms used in RIVALS

Name	Archives	Solution Concept
`Coev`	0	Maximum Expected Utility
`MinMax`	0	Best Worst Case
`MaxSolve`	2	Maximum Expected Utility
`IPCA`	1	Pareto Optimality
`rIPCA`	2	Pareto Optimality

3.2.2 Logical to Physical network

The key difference between the "logical" version and the "logical to physical" version of the simulator is that this version increases the complexity and realism by simulating messages flowing through the physical layer of the network as opposed to just through the virtual Chord overlay ring. As a result, in this version, when sending a message, instead of modeling this as the message hopping through the ring and reaching its destination, each hop from one node to another overlays the message passing from the equivalent nodes but along the routes and routers of the actual physical network.

3.3 Coevolutionary Algorithms with Archives

In RIVALS we use multiple solution concepts and coevolutionary algorithms with GE, see Table 1. Our baseline algorithm, named `Coev`, is a simple coevolutionary algorithm without an archive [11] that uses a maximum expected utility solution concept. When it is configured to use a best worst solution concept instead, we call it `MinMax`. A third algorithm, `MaxSolve` uses the maximum expected utility solution concept and both a *solution* and a *test* archive [5]. It manages archive growth with a hard maximum size limit. Upon reaching maximum size, it winnows the archive according to how many attacks a defender resists or vice-verse, how many defense an attack is effective against. Our fourth algorithm is Incremental Pareto-Coevolution Archive technique (`IPCA`), shown in Algorithm 1. `IPCA` uses a *solution* archive and the Pareto Optimal Set solution concept [13]. The archive is maintained by selecting *solutions* that are not dominated by other *solutions* in terms of which *tests* they solve, i.e. are useful. That is, if a *solution*, X, only solves tests A and B, and *solution*, Y, only solves test A, then *solution* X dominates Y and Y is removed from the archive. This provides monotonic evolutionary progress. Finally, our fifth algorithm is is `rIPCA`, an extension of `IPCA` [9]. `rIPCA` applies the Pareto Optimal Set solution concept to both *solution* and *test* populations, as opposed to just the *solution* population as done in `IPCA` (see ALG.1 line 9). This unfortunately erases the monotonicity property of `IPCA` but it provides memory to both adversaries, rather than just one. For more details of `rIPCA` see [9]. In both `IPCA` and `rIPCA` we consider a *solution* to be a defense and a *test* to be an attack.

Algorithm 1 `IPCA`, `rIPCA`

procedure IPCA(populations, generations)
  $t \leftarrow 0$
  $D^0 \leftarrow \text{populations}_{\text{defenders}}$ ▷ Defender is *solution*
  $A^0 \leftarrow \text{populations}_{\text{attackers}}$ ▷ Attacker is *test*
  $D^*, A^* \leftarrow \emptyset$ ▷ Best solutions
  while $t <$ generations **do** ▷ Iterate for # generations
    $A^t \leftarrow \text{NonDominated}(D^t, A^t)$ ▷ Extract attacker pareto-front
    if `rIPCA` **then**
      $D^t \leftarrow \text{NonDominated}(A^t, D^t)$ ▷ Extract defender pareto-front
    $D \leftarrow \text{GenerateDefenders}(D^t)$
    $A \leftarrow \text{GenerateAttackers}(A^t)$
    $A' \leftarrow \text{UsefulAttackers}(A, A^t, D, D^t)$ ▷ Get useful attackers
    $A^{t+1} \leftarrow A^{t+1} \cup A'$
    $D^{t+1} \leftarrow D^t$
    for $i = 1..|D|$ **do**
      if $\text{UsefulDefender}(D_i, D^{t+1}, A^{t+1})$ **then** ▷ Get useful defenders
        $D^{t+1} \leftarrow D^{t+1} \cup D_i$
    if $D^{t+1} \neq D^t$ **then**
      $t \leftarrow t + 1$
    $D^*, A^* \leftarrow \text{ExtractBest}(D^{t+1}, A^{t+1})$
  return D^*, A^* ▷ Returns best solutions found

3.4 Problems and Grammars in RIVALS

RIVALS uses grammars to facilitate the expression and exploration of attack sequences and defender strategies. The grammars are very helpful in allowing domain knowledge to be naturally expressed. The ease of use of GE currently outweighs our concern regarding the low locality of GE operators [24]. We have two central grammars which each correspond to a problem we experiment with.

3.4.1 Mobile Asset Placement

The mobile asset placement problem is to optimize the strategic placement of assets in a network. In a mission scenario we assume this optimization is determined before a mission and that the optimization only addresses assets which can feasibly be moved from one node to another or spun up at different nodes, i.e. that are "mobile".

While under the threat of node-level DDOS attack, the defense must enable a set of tasks. It does this by fielding feasible paths between the nodes that host the assets which support the tasks. A mobile asset is, for example, mobile personnel or a software application that can be served by any number of nodes. A task is, for example, the connection that allows a personnel member to use a software application. We show the concept of a task as a dashed line connecting nodes in Figure 5. Attacks are models of DDOS attacks where a variable number of specific nodes are targeted and disabled. Any disabled node is considered unreachable. Thus an attack must take down the nodes which host assets that support the tasks and consequently cause mission failure.

For example in Figure 5 there are three tasks that need to be completed using six different assets. The physical network topology of the example is shown in Figure 5a. The virtual (logical) overlay with the three tasks and assets are shown in Figure 5b. An attack that results in a failed task on the network is shown in Figure 5c.

To round out the definition of a problem it is necessary to state the fitness function of the attacker and of the defender. We state these in Section 4.2.

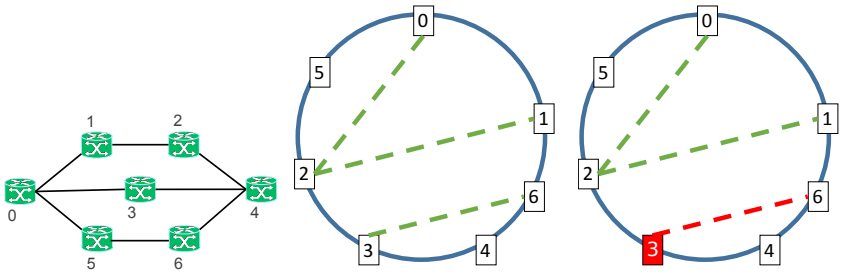
In the problem's defense grammar, each task is defined by its assets and where they are hosted. We currently assume a one-to-one mapping between assets and node identifiers, i.e. the node identifier is the same as the asset identifier.

The attack grammar for Figure 5a, Topology 0, given start symbol `<Attacks>` is:

```
<Attacks> ::= DDOSAttack(<node>)
            | DDOSAttack(<node>), <Attacks>
<node> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6
```

The corresponding grammar for the defending population with start symbol `<list>` is:

```
<list> ::= [Task1(<assets1>, <assets1>), Task2(<assets1>, <assets1>),
            Task2(<assets1>, <assets1>), Task3(<assets1>, <assets1>),
```



(a) Physical network. (b) Virtual network with 3 tasks. (c) Virtual network with 3 tasks and the node 3 attacked. The task depending on node 3 fails.

Fig. 5: Mobile asset placement problem example of physical network topology (Figure 5a), tasks on the virtual topology (Figure 5b) and an attack (Figure 5c). Dashed lines indicate the assets that are needed for the different tasks.

```
           Task5(<assets1>, <assets1>), Task6(<assets1>, <assets1>)]
<assets1> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6
```

Note that by defining different sets of assets the grammar can express constraints as to where assets can be hosted. By defining different nodes in the attack grammar, it is possible to express only the nodes reachable by the set of botnet compromised nodes. Also note that while the grammars are low level abstractions of attacks or defenses, this allows generality in the sense that they belie any number of mission or attack goals, strategies, techniques and tactics at a higher level by the attacker or defender. Finally, note that with a more complex simulator or an actual network testbed, a simple grammar change could express task ordering and dependency.

3.4.2 Network routing

The network routing problem is to complete a mission that is composed of tasks. All tasks must complete for the mission to 100% succeed. Each task is completed if source and destination nodes can be connected within a specified time interval, e.g. a message can be sent between them.

The current RIVALS attack grammar for describing the behaviors in the network routing problem is simple. An attack is one or more identifications of a node, when it will start to be attacked and the duration of the attack. Given start symbol `<Attacks>`, it is:

```
<Attacks> ::= DDOSAttack(<node>, <start_time>, <end_time>)
            | DDOSAttack(<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
```

Note that the grammar is recursive and this allows an attack to target one or more several nodes.

An example of the attack grammar used in the logical to physical version of the simulator upon receiving the start symbol `<Attacks>` is:

```
<Attacks> ::= {'physical_attacks': [<physical_attacks>],
               'logical_attacks': [<logical_attacks>]}
<physical_attacks> ::= DDOSAttack(<node>, <start_time>, <end_time>),
                         <physical_attacks>
                       | DDOSAttack(<node>, <start_time>, <end_time>)
<logical_attacks> ::= DDOSAttack(<node>, <start_time>, <end_time>),
                        <logical_attacks>
                      | DDOSAttack(<node>, <start_time>, <end_time>)
<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
```

In this grammar, because both the physical and virtual networks are utilized to represent the flow of a message, an attacker is allowed to target nodes at both the physical and virtual layers. The defense grammar in both versions of the simulator is simple because the problem assumes just three high level routing mechanisms (see Section 4.2.2). The grammar just chooses between them. This grammar, given the start symbol `<Defense>` is:

```
<Defense> ::= shortest_path_protocol
          | flooding_protocol
          | chord_protocol
```

4 Experiments

We conduct experiments using the RIVALS network simulator to demonstrate the combination of GE and coevolution for network related cybersecurity. The network simulator for our peer-to-peer network allows us to define three increasingly complex topologies and address two different problems for each of them. Each of these six combinations is what we call a *scenario*.

These experiments provide insights into how the algorithms perform as well as how they can scale over the different topologies. We present our experimental setup in Section 4.1 and scenarios in Section 4.2.

4.1 Experimental Setup

We experiment with a suite of 5 coevolutionary algorithms all with the same modular GE capability. They are presented in Table 1 and described in Section 3.3. Each experiment is one algorithm run 30 times (each time from different random initial conditions). Parameter settings for each run are presented in Table 2. Population and archive sizes reflect the search space size and time cost

Tab. 2: Coevolutionary algorithm settings for the problems. Coevolutionary algorithms specific settings are in brackets.

Parameter Setting	Value	Description
Population size	40(10 Topo 2)	number of individuals in each population
Archive size	20	max archive size (MaxSolve)
Generations	20	number of times populations are evaluated
Max length	20	max length of individual integer string
Parent archive probability	0.9	probability of choosing parent from archive (MaxSolve)
Crossover probability	0.8	probability of combining two individual integer strings
Mutation probability	0.1	probability of integer change in individual

of running the network simulator. Other parameters are standard. We present results that are averaged over the 30 runs.

We perform our tests on an Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz processor with 24 cores with 96GB of RAM. Tests are performed serially for greater accuracy and to eliminate any possible interference between tests. We report the execution time and the fitness of the best defender at the last generation as the final performance.

4.2 Scenarios

Each network simulation or "run" explores one of 6 *scenarios*. A scenario is defined by a network topology and a problem. A problem is defined by objectives for the defender and attacker, their behaviors, which are expressed by grammars, and their fitness functions.

4.2.1 Network Topologies

The experimental topologies range in size and complexity. In order to keep the network simulation simple, we assume that every edge is unit-length.

Topology 0 We start with a simple topology, see Figure 5a, that functions as a benchmark allowing us to explore simple mission scenarios exhaustively before scaling up to larger and more realistic topologies.

Topology 1 See Figure 6. This topology has 25 nodes, arranged in 4 subnets with 4 nodes conceptually functioning as fully connected subnet routers. All 25 nodes are mapped to the logical peer-to-peer ring. Topologies 1 and 2 are assumed to be too large to conveniently enumerate all the combinations of attacks and defenses.

Topology 2 See Figure 7. This topology has 36 nodes modeling subnet routers placed across the continental USA. All 36 nodes are mapped to the logical peer-to-peer ring with an assumption that they serve sub nets that are not on the ring.

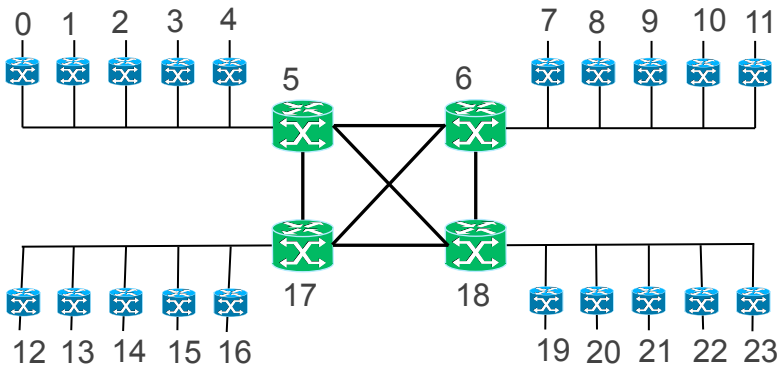


Fig. 6: Topology 1, larger network providing more nodes and a different topology

4.2.2 Problems

Each scenario solves one of two problems.

1. **Mobile Asset Placement** This problem deals with the placement of network assets to serve tasks that support a mission. Optimal placement of the assets will minimize connectivity loss in an encounter with an attack. See Section 3.4.1 for the grammars and further details. The current version of the problem has 6 tasks. Each task uses two assets.

   Attacker Fitness Function

   $$f_a^{MAP} = \frac{n\_failed}{n\_tasks} - C\frac{n\_attacks}{n\_tasks}$$

   n_tasks is the total number of tasks, n_failed is defined as how many tasks the attacker was able to disrupt, and $n\_attacks$ is the number of attacks the attacker used, $C = 1/1000$. This incentivizes attackers to disrupt tasks but with as few attacks as possible.

   Defender Fitness Function

   $$f_d^{MAP} = \frac{n\_successful}{n\_tasks} - n\_same\_nodes - n\_duplicate\_tasks$$

   n_tasks is the same as before, $n\_successful$ is defined as the total successful tasks, n_same_nodes is the number of tasks such that the start and end node are the same (path to self), $n\_duplicate\_tasks$ is the number of duplicated tasks. This incentivizes defenders to succeed at as many tasks as possible while penalizing approaches that use trivial tasks (same start and end node) or duplicate tasks.

2. **Network Routing** This problem defines a mission as the completion of tasks that successfully send a message between source and destination

Fig. 7: Topology 2, possible network for a more realistic mission.

nodes within a specified time interval. Tasks represent different elements of a mission, 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. See Section 3.4.2 for the grammars.

Network Routing can be solved with either a "logical" or "logical and physical" network simulation. This does not change the fitness function of the defender but it does change that of the attacker. We first describe the defender's routing protocol choices and fitness function. Then we describe the attacker's fitness function assuming a "logical" network simulation where all hops occur just on the logical ring network. We then explain the difference between the two types of simulations and consequently provide the attacker's fitness function for the "logical and physical" simulations.

Defender Routing Protocols The defender chooses among 3 different routing protocols that use shortest path, flooding or Chord's finger tables.

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 stabilization when a node is lost or returns to service (see 3.1).

Defender Fitness Function We reward defenders that complete the mission quickly and with few hops and punish those that take longer and use more hops. 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. The fitness function for the defender is

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

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.

Attacker Fitness Function on Logical Network: 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

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

where $mission\_success$ describes 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.

Attacker Fitness Function on Logical to Physical Network: This type of simulation is only relevant to routing with the Chord protocol. If a node finds cannot make a hop to another node in the ring,

it scans its finger table to find the node produces the largest hop and is available. How we reward attackers changes because both logical nodes and physical nodes are available for attack. Given a definition that n_nodes = n_physical + n_logical, the new fitness function for attackers is:

$$f_a^{PL} = \frac{1 - mission\_success}{n\_nodes + (2 \cdot n\_physical \cdot p\_duration) + (n\_logical \cdot l\_duration)}$$

In this equation, *mission_success* still represents whether the mission succeeded (1) or failed (0), *n_physical* describes the number of attacks launched on nodes in the physical layer, *n_logical* represents the number of attacks launched on nodes in the virtual layer, *p_duration* represents the total aggregated time nodes in the physical layer were under attack, and *p_logical* represents the total aggregated time nodes in the logical layer were under attack. This fitness function penalizes attacks launched on the physical layer more heavily because taking out a node in the physical layer can require more effort than taking out the corresponding node in the virtual layer.

5 Results

Our first question compares the algorithms. In terms of the performance (best defender fitness) and execution time of the different ones, is the `rIPCA` algorithm an improvement upon them? Since it builds upon `IPCA` by using the same archive maintenance strategy for the *solutions* as `IPCA` uses for the *tests*, can it evolve better solutions? Or, because the monotonic progress guarantee of `IPCA`'s archive is displaced by the second archive, will `rIPCA` evolve comparable or worse solutions? `IPCA`'s archive keeps every solution (to guarantee monotonic fitness trajectory). Can the cost of monotonic progress be simultaneously lowered without significant loss of performance?

Our second question is specific to the network routing problem. In terms of the algorithms' performance, would each of them be able to consistently and correctly identify the Chord protocol implementation as the network defense mechanism that is best able to handle network attacks?

5.0.1 Mobile Asset Placement

We collected timing and performance results for the mobile asset placement problem. In Table 3 we show the averaged results over 30 runs for each topology and algorithm. We see that `rIPCA` has high variance and that `IPCA` and `rIPCA` both yield high performance with `rIPCA`'s being slightly lower. We observe that `IPCA` has a significantly longest execution time in all topologies, however as the network size grows performance becomes more similar.

Tab. 3: Mobile asset placement execution time and final defender fitness (averaged over 30 runs).

Topology 0		
Algorithm	**Exec Time(s)**	**Final Perf.**
`Coev`	10.616 ± 0.444	0.132 ± 0.042
`MinMax`	8.603 ± 1.511	0.017 ± 0.050
`MaxSolve`	11.256 ± 0.507	0.282 ± 0.067
`IPCA`	24.661 ± 1.855	0.461 ± 0.069
`rIPCA`	8.079 ± 0.967	0.333 ± 0.166
Topology 1		
Algorithm	**Exec Time(s)**	**Final Perf.**
`Coev`	6.092 ± 1.249	0.380 ± 0.154
`MinMax`	4.213 ± 0.369	0.267 ± 0.200
`MaxSolve`	8.327 ± 0.429	0.267 ± 0.200
`IPCA`	12.990 ± 1.563	0.805 ± 0.063
`rIPCA`	5.932 ± 0.950	0.695 ± 0.259
Topology 2		
Algorithm	**Exec Time(s)**	**Final Perf.**
`Coev`	1.784 ± 0.328	0.182 ± 0.074
`MinMax`	1.482 ± 0.157	0.150 ± 0.094
`MaxSolve`	2.280 ± 0.127	0.184 ± 0.069
`IPCA`	4.188 ± 0.276	0.338 ± 0.074
`rIPCA`	2.245 ± 0.394	0.276 ± 0.132

Next, in Figure 8, we show how each algorithm progresses over time on Topology 0. IPCA's trajectory shows its expected monotonic increasing performance and also has the highest average final performance. `rIPCA`, while not the best algorithm, is second in average final performance while consistently performing better in execution time (refer to Table 3).

5.0.2 Network Routing: Logical Simulation

Prior to running the experiments with the network routing problem, we exhaustively searched Topology 0 (Figure 5a) for a solution. To do this, we set attacks to last the full duration of a task. We saw that *Chord* only fails if all the nodes are blocked. *Shortest path* fails if any node on the shortest path is blocked. *Flooding* fails if a start node is blocked, or an end node is blocked, or when all paths include a node that is blocked. This information provided us with a baseline of comparison outside the algorithms and it allowed us to verify algorithm correctness. It also points out that exhaustive search is possible in topologies with a small number of nodes and becomes increasingly difficult for a topology as large as the ones in Figures 6 and 7.

We run the network routing mission simulation over 30 runs and then collect the average over the results. In Table 4 we show the average and standard deviation of both the wall-clock execution times as well as of the best fitness values per generation. We first consider Topology 0. The algorithms show different results, with `IPCA` and `rIPCA` showing superior average final performance. We conjecture this is due to the test archives for both `IPCA` and `rIPCA` as these archives help enforce monotonic performance increases. When looking

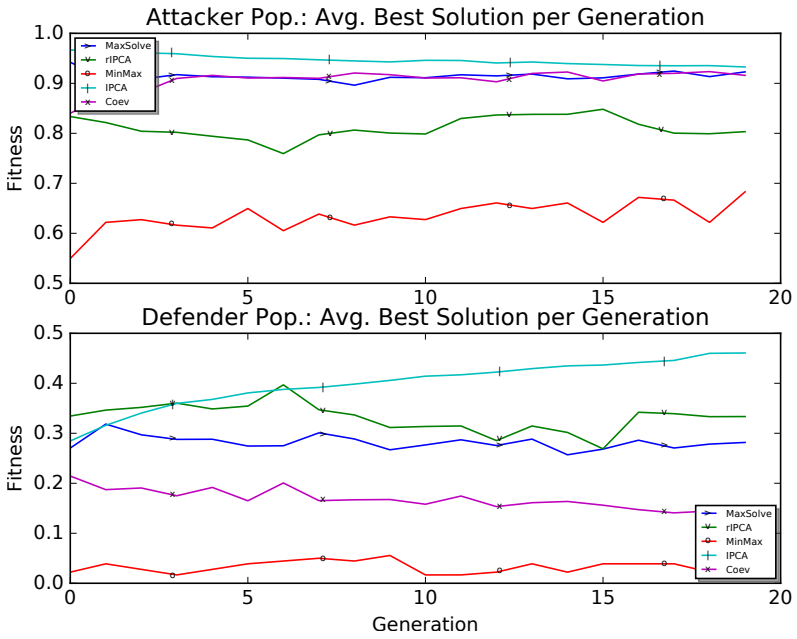


Fig. 8: Best fitness value average (over 30 runs) per generation for 20 generations on the mobile asset placement problem. Algorithms compared: `IPCA`, `rIPCA`, `Coev`, `MinMax`, `MaxSolve`.

Tab. 4: Network routing execution time and final defender fitness on logical topology (averaged over 30 runs).

Topology 0

Algorithm	Exec Time(s)	Final Perf.
Coev	10.417 ± 1.650	0.091 ± 0.014
MinMax	9.802 ± 1.693	0.045 ± 0.028
MaxSolve	20.945 ± 1.336	0.088 ± 0.022
IPCA	66.576 ± 6.537	0.097 ± 0.021
`rIPCA`	47.754 ± 8.108	0.128 ± 0.055

Topology 1

Algorithm	Exec Time(s)	Final Perf.
Coev	36.911 ± 13.290	0.008 ± 0.001
MinMax	34.745 ± 10.351	0.005 ± 0.002
MaxSolve	12.322 ± 19.236	0.007 ± 0.001
IPCA	266.382 ± 59.253	0.008 ± 0.000
`rIPCA`	267.784 ± 69.347	0.008 ± 0.000

Topology 2

Algorithm	Exec Time(s)	Final Perf.
`Coev`	180.114 ± 80.664	0.005 ± 0.000
`MinMax`	158.955 ± 72.101	0.004 ± 0.001
`MaxSolve`	768.817 ± 342.642	0.005 ± 0.001
`IPCA`	1729.165 ± 623.941	0.005 ± 0.000
`rIPCA`	1566.194 ± 643.867	0.005 ± 0.000

at Topologies 1 and 2, we do not see much difference between the algorithms. This is due to the fact that the topologies are much larger in this case and the defenses are not as versatile. However, `rIPCA` is on par or better than `IPCA` in terms of execution time.

In Figure 9, we examine the average fitness values for both attack and defense populations from one `Coev` run over Topology 0. In the attacker's average fitness plot, the average fitness for the attack population experiences a short increase in performance then quickly drops to 0. It then oscillates as the defense population converges on Chord. The variation in the algorithm allows non-optimal (i.e. non-chord) solutions to form part of the defense population. This, in turn, increases the average fitness of attacks as they face defenses which they can succeed against.

The network routing mission experiments show that `IPCA` and `rIPCA` perform better 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 yield as strong defenders as for a fixed attack [9].

5.0.3 Network Routing – Physical and Logical Simulation

In these experiments, see Table 5, the trends in the defender fitness values across the topologies and algorithms closely resemble the trends we noted in Table 4. The difference, however, upon inspecting the outputs of the algorithms, is that rather than algorithms converging to the Chord protocol as the best

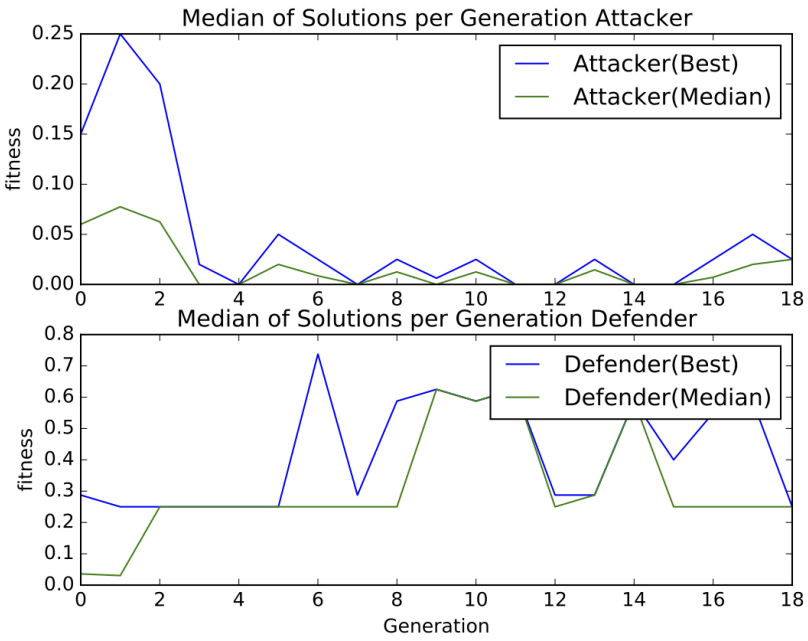


Fig. 9: Results from a `Coev` run for network routing on logical topology on network topology 0. Top: Median and best fitness results for attacker population over 20 generations. Bottom: Median and best fitness results for defender population over 20 generations.

Tab. 5: Final defender fitness on each topology with physical and logical simulation (averaged over 30 runs).

	Topology 0	*Topology 1*	*Topology 2*
Algorithm	**Final Perf.**	**Final Perf.**	**Final Perf.**
`Coev`	0.079 ± 0.010	0.007 ± 0.001	0.005 ± 0.000
`MinMax`	0.053 ± 0.023	0.004 ± 0.001	0.004 ± 0.001
`MaxSolve`	0.061 ± 0.018	0.006 ± 0.001	0.002 ± 0.001
`IPCA`	0.082 ± 0.009	0.007 ± 0.000	0.005 ± 0.001
`rIPCA`	0.095 ± 0.027	0.008 ± 0.001	0.005 ± 0.001

solution for the defender, it varied in Topology 0 and Topology 1 between the Chord protocol and the flooding protocol. In the largest topology, the flooding protocol was always found by all of the algorithms as the best defender. Given these results, it is possible to recognize that increasing the complexity of the simulator to traverse the physical network between two nodes for every hop between the corresponding nodes in the virtual layer increased the number of hops the Chord protocol took to get the message to the destination. We also observed that the shortest path never evolved as a final solution. This indicated that the Chord protocol and the flooding protocol are more robust in terms of withstanding attackers.

6 Conclusions and Future Work

We have shown how to combine Grammatical Evolution and competitive coevolution so that it is possible to investigate adversarial problems and cyber arms races. In particular, we focused on network defenses and DDOS attacks. We grounded our work by considering peer to peer networks, specifically the Chord protocol, and node loss. Grammars were convenient for representing the search space of defender and attacker actions and we have embedded a GE module in each of our coevolutionary algorithms. When we use our system to solve different problems, we only have to change the BNF grammar, the interpreter and the fitness function for each problem, rather than change the genotype representation. This modularity of GE and the reusability of the GE parser and rewriter are efficient software engineering and problem solving advantages. The grammar further helps us communicate with application domain stakeholders and increases their confidence in solutions and our system.

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. 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[15]. 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 element of future work is to extend the Chord protocol. Others include: experimenting with more scenarios, generating more complex missions, e.g. with different numbers of tasks and creating

a *compendium* approach to pooling attacks and defenses from multiple runs to more explicitly choose an overall most robust defense. Finally, we will continue to improve the grammars, performance and speed of the coevolutionary algorithms.

Acknowledgements

This material is based upon work supported by DARPA. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements. Either expressed or implied of Applied Communication Services, or the US Government.

References

[1] Kamal Adamu and Steve Phelps. Coevolutionary grammatical evolution for building trading algorithms. In *Electrical Engineering and Applied Computing*, pages 311–322. Springer, 2011.

[2] Manuel Alfonseca and Soler Gil. Evolving a predator–prey ecosystem of mathematical expressions with grammatical evolution. *Complexity*, 20(3):66–83, 2015.

[3] R Muhammad Atif Azad and Conor Ryan. An examination of simultaneous evolution of grammars and solutions. In *Genetic Programming Theory and Practice III*, pages 141–158. Springer, 2006.

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

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

[6] Ian Dempsey, Michael O'Neill, and Anthony Brabazon. *Foundations in grammatical evolution for dynamic environments*, volume 194. Springer, 2009.

[7] Sevan Gregory Ficici. *Solution concepts in coevolutionary algorithms*. PhD thesis, Brandeis, 2004.

[8] Patrick Gabrielsson, Ulf Johansson, and Rikard Konig. Co-evolving online high-frequency trading strategies using grammatical evolution. In *Computational Intelligence for Financial Engineering & Economics (CIFEr), 2104 IEEE Conference on*, pages 473–480. IEEE, 2014.

[9] Dennis Garcia, Anthony Erb Lugo, Erik Hemberg, and Una-May O'Reilly. Investigating coevolutionary archive based genetic algorithms on cyber defense networks. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1455–1462. ACM, 2017.

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

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

[12] Malcolm I Heywood. Evolutionary model building under streaming data for classification tasks: opportunities and challenges. *Genetic Programming and Evolvable Machines*, 16(3):283–326, 2015.

[13] Edwin D de Jong. A monotonic archive for pareto-coevolution. *Evolutionary Computation*, 15(1):61–93, 2007.

[14] Krzysztof Krawiec and Malcolm Heywood. Solving complex problems with coevolutionary algorithms. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 687–713. ACM, 2016.

[15] Aleksandar Kuzmanovic and Edward W Knightly. 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*, pages 75–86. ACM, 2003.

[16] Claire Le Goues, Anh Nguyen-Tuong, Hao Chen, Jack W Davidson, Stephanie Forrest, Jason D Hiser, John C Knight, and Matthew Van Gundy. Moving target defenses in the helix self-regenerative architecture. In *Moving Target Defense II*, pages 117–149. Springer, 2013.

[17] Andrew R McIntyre and Malcolm I Heywood. Multi-objective competitive coevolution for efficient gp classifier problem decomposition. In *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, pages 1930–1937. IEEE, 2007.

[18] Andrew R McIntyre and Malcolm I Heywood. Cooperative problem decomposition in pareto competitive classifier models of coevolution. In *Genetic Programming*, pages 289–300. Springer, 2008.

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

[20] George Rush, Daniel R Tauritz, and Alexander D Kent. Coevolutionary agent-based network defense lightweight event system (candles). In *Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference*, pages 859–866. ACM, 2015.

[21] Kenneth O Stanley and Risto Miikkulainen. Competitive coevolution through evolutionary complexification. *J. Artif. Intell. Res.(JAIR)*, 21:63–100, 2004.

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

[23] Daniel R Tauritz et al. A no-free-lunch framework for coevolution. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 371–378. ACM, 2008.

[24] Peter A Whigham, Grant Dick, James Maclaurin, and Caitlin A Owen. Examining the best of both worlds of grammatical evolution. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, pages 1111–1118. ACM, 2015.

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