Coevolutionary Genetic Algorithms for Proactive Computer Network Defenses

by

Anthony Erb Lugo

S.B., Computer Science and Mathematics M.I.T., 2016

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2017

Author ....
Department of Electrical Engineering and Computer Science
May 26, 2017

Certified by..
Una-May O'Reilly
Principal Research Scientist
Thesis Supervisor

Certified by..
Erik Hemberg
Research Scientist
Thesis Supervisor

Accepted by ....
Christopher Terman
Chairman, Masters of Engineering Thesis Committee

Coevolutionary Genetic Algorithms for Proactive Computer Network Defenses

by

Anthony Erb Lugo

Submitted to the Department of Electrical Engineering and Computer Science
on May 26, 2017, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis explores the use of coevolutionary genetic algorithms as tools in developing proactive computer network defenses. We also introduce rIPCA, a new coevolutionary algorithm with a focus on speed and performance. This work is in response to the threat of disruption that computer networks face by adaptive attackers. Our challenge is to improve network defenses by modeling adaptive attacker behavior and predicting attacks so that we may proactively defend against them. To address this, we introduce RIVALS, a new cybersecurity project developed to use coevolutionary algorithms to better defend against adaptive adversarial agents. In this contribution we describe RIVALS' current suite of coevolutionary algorithms and how they explore archiving as a means of maintaining progressive exploration. Our model also allows us to explore the connectivity of a network under an adversarial threat model. To examine the suite's effectiveness, for each algorithm we execute a standard coevolutionary benchmark (Compare-on-one) and RIVALS simulations on 3 different network topologies. Our experiments show that existing algorithms either sacrifice execution speed or forgo the assurance of consistent results. rIPCA, our adaptation of IPCA, is able to consistently produce high quality results, albeit with weakened guarantees, without sacrificing speed.

Thesis Supervisor: Una-May O'Reilly
Title: Principal Research Scientist

Thesis Supervisor: Erik Hemberg
Title: Research Scientist

Acknowledgments

I would like to thank Erik Hemberg and Una-May O'Reilly for giving me the opportunity to work on this project. Also, in particular, for their substancial support and guidance throughout this research project. Throughout this project I have grown and have learned a great deal. Without their help, this project would not have been accomplished.

I also would like to thank Dennis García, fellow MEng student and collaborator on the RIVALS project, for always being available to provide support when needed.

To my friends, thank you for all the great times and for always being there through the toughest of times.

Finally, I would like to thank my family for their never ending support and guidance. It is because of them that I am where I am today.

Contents

List of Figures

List of Tables

Chapter 1

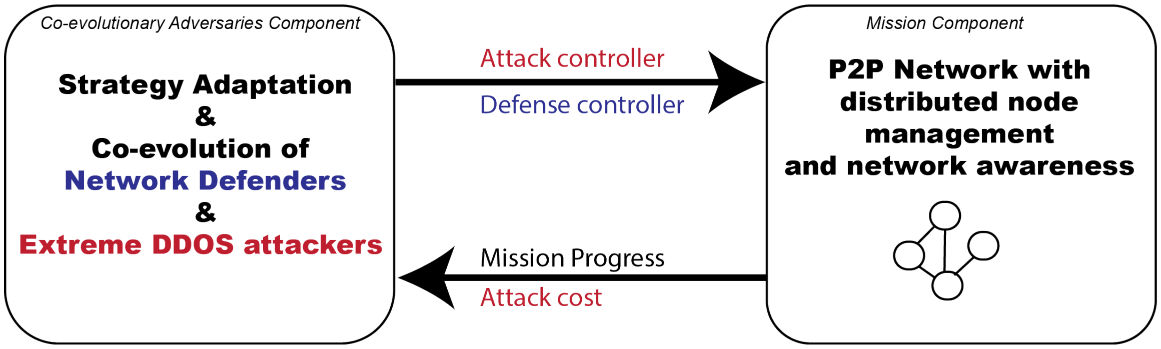
Introduction

1.1 Motivation

Along with the steady increase of internet connected devices in the past decade, we, as a society, have faced an ever growing list of cyber attacks with ever increasing sophistication. As we rely more and more on networks to handle our critical or sensitive information, it is necessary to make sure that we proactively maintain and update the security of our networks. Unfortunately, today, many networks are implemented with cybersecurity as a reactive action rather than a proactive one. Particularly, they lack the capability to predict potential advanced adaptive attacks. That is, when an attack is mitigated by a specific defense, it is likely that the attacker will adapt their attack to bypass the updated network defenses. The network defenses then have to update as well in response to the attacker's new attack. This sequence continues and causes both parties to evolve in response to each other's actions.

This arms race or attacker-defender dynamics is similar to coevolution in biology. As such, this problem is well suited to being studied using coevolutionary algorithms. By leveraging this class of algorithms, we hope to improve the landscape for defenders as they can then use them to prepare better defenses by simulating an attacker's behavior on the network.

In simulating any type of complex behavior with evolutionary algorithms, we face unique issues. Since simulation can be costly, speed of execution and efficiency of



Co-evolutionary Adversaries Component
Strategy Adaptation & Co-evolution of Network Defenders & Extreme DDOS attackers
Attack controller
Defense controller
Mission Component
P2P Network with distributed node management and network awareness
Mission Progress
Attack cost

Figure 1-1: RIVALS system overview.

the algorithm become increasingly important. We must also ensure that the results returned by the algorithm are effective and useful. With these issues in mind, it is necessary to explore the trade-offs provided by different algorithms as they respect to speed and quality of results as well as algorithm objectives. Other trade-offs include complexity of fitness calculation, goal of fitness evaluator and population archives among others. Fitness in this case relates to the fitness of an individual as it compares to the adversarial population (i.e. if an individual can deter an attack, its fitness is higher). As an example of a trade-off made, while archives may provide monotonic increasing performance for a given population, they come at a high cost on execution time. Thus, an otherwise powerful algorithm can be potentially made unfit for many real-world applications.

This research forms part of a larger project, RIVALS (Garcia et al., 2017), a new cybersecurity project that takes advantage of coevolutionary algorithms. RIVALS makes use of coevolutionary algorithms in order to return the optimal defenses available when under the threat of an adaptive adversary, see Figure 1-1. RIVALS' purpose is to ensure that we are able to provide resilient network configurations that can sustain adaptive attacks with respect to network missions or network connectivity. Through these analyses we can help network designers improve their networks with proactive attackers in mind.

1.2 Research Question

Is it possible to effectively employ coevolutionary algorithms to appropriately model the complex adaptive behavior of attackers and defenders? Moreover, since this behavior is inherently complex and network simulation requires a large amount of computation, we want to know: can we improve on existing algorithms by applying trade-offs in favor of speed while perhaps sacrificing performance guarantees?

1.3 Contributions

We analyze the effectiveness of our suite of coevolutionary algorithms on both Compare-on-one, a standard coevolutionary algorithm baseline problem, and on our network simulator, RIVALS. With Compare-on-one, we can compare our implementations with previous results and also provide benchmarks for our own variations of the algorithms. Additionally, we introduce rIPCA, a modified version of IPCA in which we expand on the idea of non-domination and apply this concept to both populations. This variation implies that we now have weaker guarantees, however, in practice, despite this trade-off, we find a much faster algorithm with only a slight drop in the quality of results. Lastly, we determine the usefulness of rIPCA and the other coevolutionary algorithms with respect to our adversarial network simulator. Our network simulator measures the performance of these algorithms by applying them to two different network security problems: strategic network resource placement and network mission completion. We have configured these problems to work in conjuntion with our suite of coevolutionary algorithms. The results and analysis of our experiments are presented in Chapter 4.

Chapter 2

Related Work

Coevolutionary algorithms such as the IPCA (Jong, 2007) algorithm allow us to recreate coevolutionary behavior programmatically. Here we introduce existing work that considers applications of coevolutionoary algorithms. We also describe work related to cyber security and how previous research has explored adversarial behavior.

2.1 Coevolution

This research is inspired by *STEALTH (Simulating Tax Evasion And Law Through Heuristics)* (Hemberg et al., 2016), a paper which applies coevolutionary algorithms to the field of tax law in order to detect workarounds to tax policy. We build upon the fact that the adversarial setting between tax evaders and tax policy writers follows the same type of adversarial dynamics between network attacker and network defenders. As such, we are able to apply some of the same coevolutionary concepts to our field of study, cybersecurity.

We also see coevolution in an adversarial setting in “Increasing infrastructure resilience through competitive coevolution” (Service and Tauritz, 2009). This paper explores coevolution as a means to improve infrastructure resilience. In “Red teaming with coevolution” (Hingston and Preuss, 2011), the authors present coevolutionary algorithms as a means to determine "strategic and tactical options available to each side in a conflict situation". These papers are similar to ours in that they deal with coevolution in a competitive setting within an adversarial environment. Our work on RIVALS, however, focuses on adversarial behavior within network security.

2.2 Adversarial Cyber Security

The study of adversarial dynamics, as it relates to cybersecurity, has expanded recently within the field of Artificial Intelligence. As in our published work Garcia et al., 2017, we see examples of study of adversarial dynamics in the fields of Evolutionary Computation, Machine Learning, Game Theory and AI-planning. We now introduce examples of research in these fields in their respective order. In Haddadi and Zincir-Heywood, 2015, the authors utilize Evolutionary Computation to analyze botnet dection systems and the effect of botnet evolution. With regards to Machine Learning, we see how patterns learned from data gathered can be useful in email spam-filters (Dalvi et al., 2004). Next, Game Theory, a powerful tool used to determine optimal outcomes in situations between conflicting parties, has seen applications in adversarial behavior through the paper "autonomous, collaborative control for resilient cyber defense" Wagner et al., 2012. Lastly, in AI-Planning, where programs develop a strategy according to a given set of goals, we see considerations of adversarial dynamics in Silva Arantes et al., 2015. The aforementioned paper considers "UAV Path Re-planning" under critical situations.

Within the cybersecurity field we also see adversarial dynamics being studied through the study of Moving Target Defenses (MTDs). The goal of employing MTDs is to force an adversary to periodically replan its attack as the defender re-allocates its resources. Previous research (Winterrose and Carter, 2014) has investigated the use of genetic algorithms in evolving attackers in an environment where defenders employ the MTD technique.

2.3 Adversarial Coevolutionary Cyber Security

Combining these two topics, Coevolutionary Agent-based Network Defense Lightweight Event System (CANDLES) (Rush, Tauritz, and Kent, 2015), presents a system that coevolves agent defenders and attackers. Bearing similarity with RIVALS, CANDLES uses coevolutionary genetic algorithms. However, RIVALS and CANDLES differ in that RIVALS presents a much more concrete simulation.

While previous research has explored both the field of Coevolution as well as the field of Adversarial Cyber Security, we have noted that there is currently a lack in research related to the intersection of these topics. Our research aims to reduce this gap through our analysis of coevolutionary algorithms in the context of network security, with a focus on developing defenses against adaptive adversarials. Specifically, adversarials with the power to perform DOS (Denial-of-Service) attacks that can disrupt a network or effectively take down a server and which aim to bypass defensive measures. This work, together with the other components of **RIVALS**, has been shown to be novel and relevant to both the field of coevolution and the field of cybersecurity (Garcia et al., 2017).

Chapter 3

Method

In this chapter we introduce our suite of coevolutionary algorithms along with an overview of the grammars used to incorporate with RIVALS.

3.1 Coevolutionary Algorithms

Coevolutionary algorithms provide a means with which to model coevolutionary behavior according to a **solution concept** (Popovici et al., 2012). Where a **solution concept** represents a heuristic for selecting the best individuals per population over a given generation. In particular, solution concepts may focus on optimizing the fitness (quality) of individuals for a specific population or they may weigh each population equally. The quality, or fitness, of an individual is determined by how that individual performs against its adversaries, the individuals in the adversarial population. These adversaries are called *tests* while the individual being optimized is called a *solution*.

3.1.1 Solution Concepts

A solution concept provides a heuristic for solutions in an evolutionary context. This notion is important as different problems may require different types of solutions. As such, it is important that we be able to measure the quality of a solutions (i.e. its fitness) according to a specific goal. Examples of these solution concepts include (Garcia et al., 2017):

- **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
- **Maximization of Expected Utility** A solution's fitness reflects that its tests are of equal importance.
- **Nash Equilibrium** favors solutions which lead to stable solution states in which no sole actor can their improve their state unilaterally.
- **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.

3.1.2 Coevolutionary Algorithms with Archives

Pathologies arise in coevolutionary optimization due to its complex dynamics, (Krawiec and Heywood, 2016). These include (Garcia et al., 2017):

- Intransitivity, e.g.
  - Red Queen Effect
  - Cycling
  - Transitive dominance, and,
- Disengagement (loss of gradient), e.g.
  - *solution* fails to perform in any way on a *test*
  - inability to discover a *test* to efficiently search for *solutions*.

To mitigate some of these issues, archives can be added into the coevolution process so as to maintain a history of useful solutions with the goal of preserving known good solutions. An archive maintains these solutions apart from the main coevolutionary process so that they may be stored and not lost (Krawiec and Heywood, 2016; De Jong, 2005; Jong, 2007; Liskowski and Krawiec, 2016). A coevolutionary algorithm may employ these archives on one or more of the populations it maintains.

We now describe the suite of coevolutionary algorithms we have implemented for experimentation with RIVALS (Garcia et al., 2017):

1. `COEV` and `MinMax`(Algorithm 1): presents a simple coevolutionary algorithm (Hemberg et al., 2016). As seen in the pseudocode, we can adapt its fitness evaluation to use either the maximum expected utility solution concept or the best worst solution concept (MinMax).

2. IPCA and rIPCA (Algorithm 2): presents our implementation of the IPCA algorithm which makes use of 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).

3. MaxSolve(Algorithm 3): uses the maximum expected utility solution concept and archives (De Jong, 2005).

Algorithm 1 `Coev`

```
 1: procedure COEV(populations, generations)
 2:     $t \leftarrow 0$
 3:     best_individuals $\leftarrow \emptyset$
 4:     while $t <$ generations do                          ▷ run for # generations
 5:         pop$' \leftarrow$ Generate(populations)
 6:         if BestWorstCase then
 7:             pop$' \leftarrow$ EvalBestWorstCaseFitness(pop$'$)
 8:         if MaximumExpectedUtility then
 9:             pop$' \leftarrow$ EvaluateMEUFitness(pop$'$)
10:         populations $\leftarrow$ Merge(populations, pop$'$)
11:         populations $\leftarrow$ SortPopulations(populations)
12:         best_individuals $\leftarrow$ ExtractBest(populations)
13:         $t \leftarrow t + 1$
14:     return best_individuals                        ▷ Returns best solutions found
```

Algorithm 2 IPCA, rIPCA

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

Algorithm 3 MaxSolve

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

3.1.3 rIPCA

Our proposed variation of IPCA, rIPCA, follows the main structure of the original algorithm, however, our addition to filter out dominated *test* individuals means that we no longer keep all useful tests. As such, we conjecture that rIPCA loses this quality and is therefore at a disadvantage with respect to IPCA in terms of the quality of solutions that it is able to produce. In contrast, our changes allow our algorithm to still perform well on our problem test suite, regardless of its guarantees, while also improving the simulation execution time drastically. Since IPCA retains all of the useful individuals from one population, the population size rises indefinitely and forces the algorithm to perform many more calculations than perhaps are necessary. rIPCA, on the other hand, filters both the defending population as well as the attacking population. Thus, we are able remove a large portion of the computation required by IPCA.

3.2 Grammatical representation for coevolutionary search

RIVALS (Garcia et al., 2017) uses 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 (O'Neill and Ryan, 2003). An example of GE and competitive coevolution is in the investigation of spatial coevolution of age layered planes in robocode (Harper, 2014). The ease of use of GE currently outweighs our concern regarding the low locality of GE operators, e.g. (Whigham et al., 2015).

We have two main grammar classes for our experiments. The first, to optimize for strategic resource placement, is our network placement grammar class. In this grammar, attacks are modeled as simple DOS (Denial of Service) attacks in which a targeted node is considered unreachable for the entirety of its evaluation. The defenses are set as tasks which attempt to find a path between two nodes. The attack's fitness will improve if it can cause more tasks to fail. The attack grammar for Topology 0, given start symbol `<Attacks>`, can be expressed as:

$\langle Attacks \rangle$ `::= DOSAttack(`$\langle node \rangle$`)`

`|  DOSAttack(`$\langle node \rangle$`),` $\langle Attacks \rangle$

$\langle node \rangle$ `::= 0 | 1 | 2 | 3 | 4 | 5 | 6`

We note that this grammar is recursive with respect to the `<Attacks>` symbol. This allows for more variation within the attacker population. Our fitness function, however, will take into account the amount of attacks included when determining the performance of that attack, favoring those attacks which minimize their use of resources. The corresponding grammar for the defending population with start symbol `<list>` is:

$\langle list \rangle$ `::= [Task(`$\langle node \rangle$`,` $\langle node \rangle$`), Task(`$\langle node \rangle$`,` $\langle node \rangle$`), Task(`$\langle node \rangle$`,` $\langle node \rangle$`), Task(`$\langle node \rangle$`,` $\langle node \rangle$`), Task(`$\langle node \rangle$`,` $\langle node \rangle$`), Task(`$\langle node \rangle$`,` $\langle node \rangle$`)]`

$\langle node \rangle$ `::= 0 | 1 | 2 | 3 | 4 | 5 | 6`

This grammar denotes that each defense will test the connectivity between six pairs of nodes in the network through our concept of a Task.

While resource placement is a useful problem, we also care about the capability to carry out network tasks even if a node is unreachable for a certain period of time. For this case, we introduce our second class of grammars, the network mission grammars, which are time-aware. The corresponding attack grammar is as follows:

$\langle \mathit{Attacks} \rangle$ `::= DOSAttack(`$\langle \mathit{node} \rangle$`,` $\langle \mathit{start\_time} \rangle$`,` $\langle \mathit{end\_time} \rangle$`)`

`|  DOSAttack(`$\langle \mathit{node} \rangle$`,` $\langle \mathit{start\_time} \rangle$`,` $\langle \mathit{end\_time} \rangle$`),` $\langle \mathit{Attacks} \rangle$

$\langle \mathit{node} \rangle$ `::= 0 | 1 | 2 | 3 | 4 | 5 | 6`

$\langle \mathit{start\_time} \rangle$ `::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`

$\langle \mathit{end\_time} \rangle$ `::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`

The defending grammar for the network missions problem makes use of three different routing protocols. This grammar is simple as it is limited to only a choice between three options and is thus not presented here. Lastly, as we have three different topologies for our RIVALS simulator, we implement three versions of these grammars, each with the corresponding node and time values.

Chapter 4

Experiments

This chapter presents the experiments we ran on our suite of coevolutionary algorithms. We describe our configurations for the algorithms implemented, include our results and then follow with a discussion of the results.

4.1 Setup

Our experiments include running on Compare-on-one, a standard coevolutionary algorithm benchmark, as well as experiments using the RIVALS network simulator. The network simulator has been set up with three increasingly complex topologies as well as two different optimization problems. Strategic network resource placement, our first optimization problem, deals with placement of network resources so as to minimize potential loss of connectivity when faced against an adaptive adversarial. The second problem, network defenses of mission critical tasks, deals with optimizing the defense strategy (in our case, network protocol) when faced against an adaptive adversarial. These experiments help bring us insights into how the algorithms perform as well as how they can scale over the different topologies.

In Table 4.1 we include our configuration of the algorithms. Our settings follow the settings for Compare-on-one used in IPCA (Jong, 2007) so that we may compare with previous results and similar works. Our network configuration follows that of our published work (Garcia et al., 2017). Each experiment is run over 30 iterations

Table 4.1: Algorithm Settings

Parameter Setting	Compare-on-one	Network Simulations & Network Resource Placement
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
Grammar	No	Yes

and the results are averaged to produce the final results.

We perform our tests on a 24 core, each core an Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz processor, machine with 96GB of RAM. Tests are performed serially for greater accuracy and to eliminate any possible interference between tests.

4.1.1 Compare-on-one

In Compare-on-one Jong, 2007, there are two populations: *learners* and *tests*. Each individual in either population is a vector of real numbers with a set size, 10 for example. When an individual, *solution* A, is compared against an individual of the opposite population, *test* B, we consider the position in B's vector which contains the highest value. If A's value in the corresponding position is higher than or equal to B's, then A is said to defeat B and lose otherwise. Performance for Compare-on-one is measured as the "lowest value among all dimensions of an individual" (Jong, 2007). As mentioned before, we run our experiments on Compare-on-one over 30 runs and average the results.

4.1.2 Network Mission Simulations

In Garcia et al., 2017 we describe the setup of our network mission simulator. We reproduce those details here for convenience:

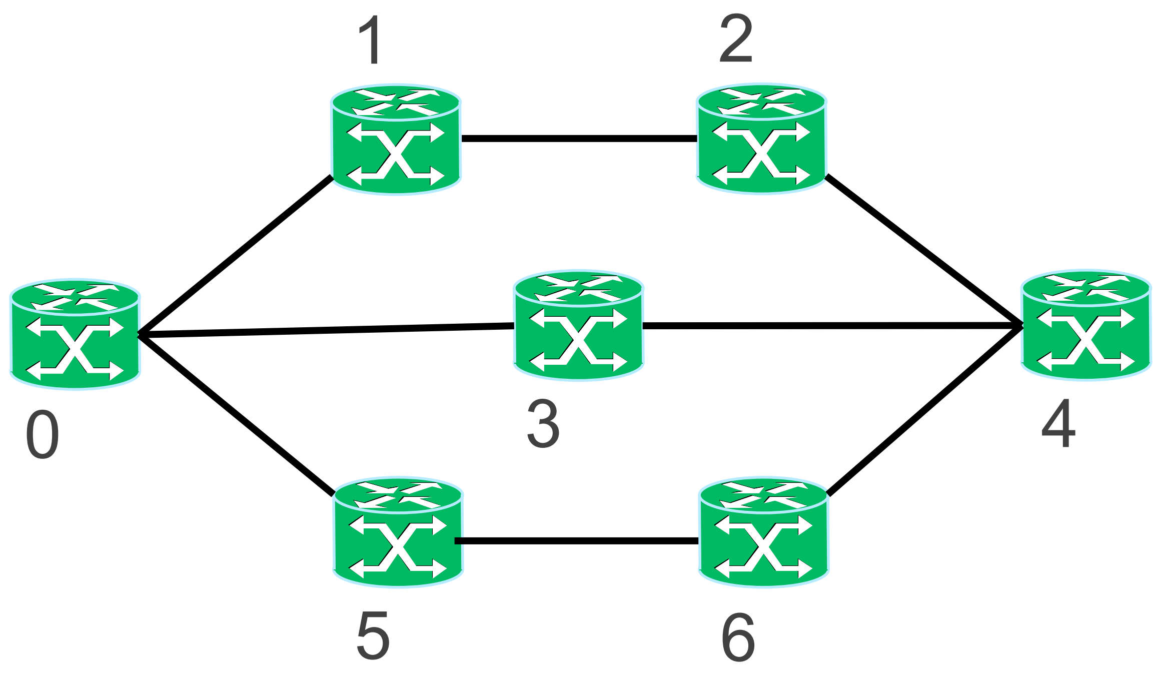


Figure 4-1: Topology 0, simple network, used to benchmark the defensive actions for routing of Shortest-path, Flooding and Chord

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, in section 3.2, allows nodes to be selected and flooded.

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

Missions: A mission is 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

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

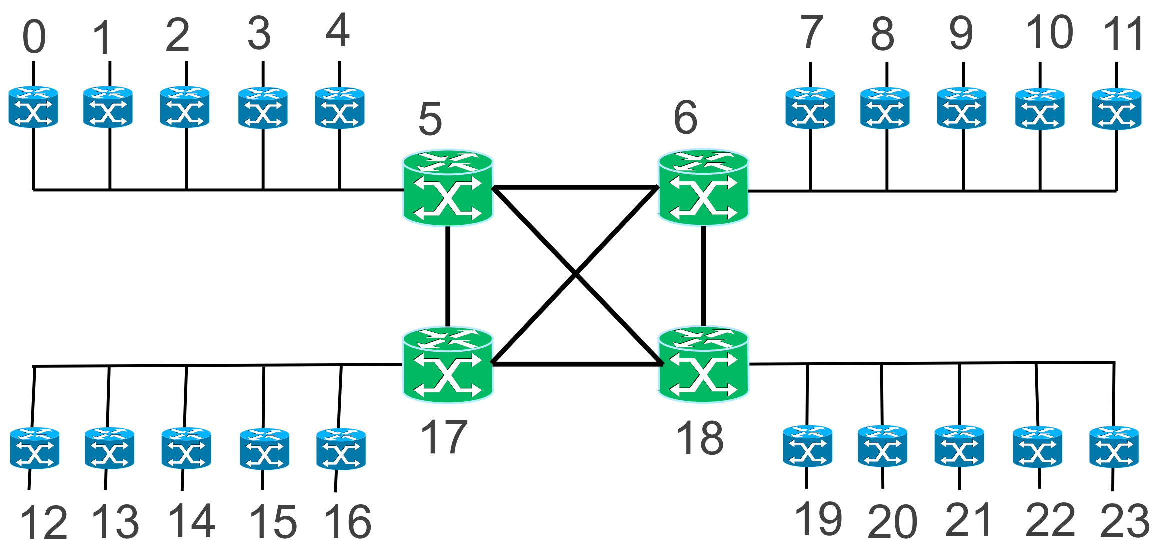


Figure 4-2: Topology 1, larger network providing more nodes and a different topology

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.

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.

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.

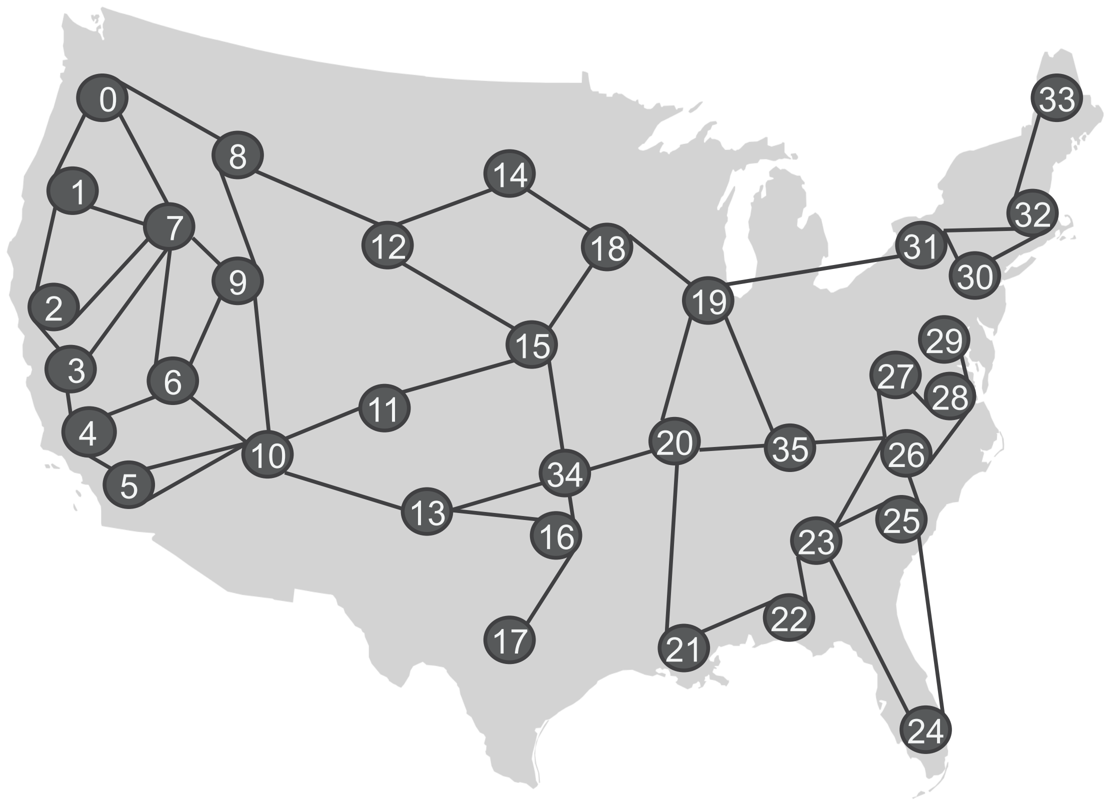


Figure 4-3: Topology 2, possible network for a more realistic mission

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 Stoica et al., 2001).

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. 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.

4.1.3 Strategic Network Resource Placement

Given a network, we would like to know where we should place our most active/critical services such that, even while under attack, services remain connected to their peers. To solve this problem we consider our problem under our coevolutionary model and assign defending and attacking population. We represent defender individuals as a series of tuples of start and end nodes in the network (see Sec. 3.2). These individuals work to determine if there is a working path between the given start and end nodes under the presence of an attack. An attacker in this problem works by selecting a few nodes to attack with DOS attacks. The selected nodes by the attacker are then set to be unreachable. The attacker's goal is to disrupt the network as much as possible while minimizing the use of resources. The goal of the defender is to select routes which maintain high connectivity even while under attack.

Given these goals for the defenders and attackers, we introduce the following fitness functions:

- **Defender**:

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

  Where n_tasks represents the number of tasks, $n\_successful$ represents the amount of successful tasks, n_same_nodes represents how many tasks are such that the task's start is the same as its end, and $n\_duplicate\_tasks$ counts how many tasks are duplicated. This formula works to promote defenses which are successful and which don't include trivial tasks such as those represented by n_same_nodes and which also don't include duplicate tasks.

- **Attacker**:

  $$f_a = \frac{n\_failed}{n\_tasks} - \frac{n\_attacks}{1000 \cdot n\_tasks}$$

  In this case, the individual is rewarded for causing tasks to fail but is also penalized for performing many attacks, with respect to the number of tasks it faces. The variable n_failed counts the amount of failed tasks and $n\_attacks$ counts the amount of nodes this individual will attack. Similar to before, n_tasks represents the number of tasks.

As with the previous experiment setups, we run this experiment over 30 runs and average the results.

Table 4.2: Compare-on-one: Execution time and performance results (higher is better) for each of the different coevolutionary algorithms. Results are after 1000 generations and the results are averaged over 30 runs.

Algorithm	**Exec Time(s)**	**Final Perf.**
Coev	15.171 ± 1.393	0.523 ± 0.314
MinMax	12.467 ± 0.934	0.000 ± 0.000
MaxSolve	23.146 ± 0.164	0.497 ± 0.128
IPCA	935.721 ± 259.952	0.848 ± 0.045
rIPCA	61.940 ± 30.404	0.780 ± 0.228

4.2 Results

In terms of performance, 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. Our experiments show that rIPCA is able to produce similar results to those of IPCA while performing better in terms of execution time. This section now expands on each of the results from the experiments.

4.2.1 Compare-on-one

In Table 4.2 we see the results for Compare-on-one on our suite of algorithms. IPCA and rIPCA perform notably better, with IPCA having less variance and rIPCA performing much faster in terms of execution time. This is expected as IPCA maintains an archive with monotonic increasing performance while rIPCA opts to simply maintain non-dominated archives. Non-dominated archives allow rIPCA to limit the size of its test population. As such, rIPCA manages to reduce the amount of fitness evaluation calls necessary to perform well. However, as seen in Figure 4-4, when running IPCA and rIPCA for 1250 generations, IPCA is seen to outpace rIPCA in terms of the quality of results it produces for the learner (defending) population. Unfortunately, the execution time trade-off for this level of performance would render IPCA too costly for applicable use in RIVALS. Thus making rIPCA a strong contender for our RIVALS system.

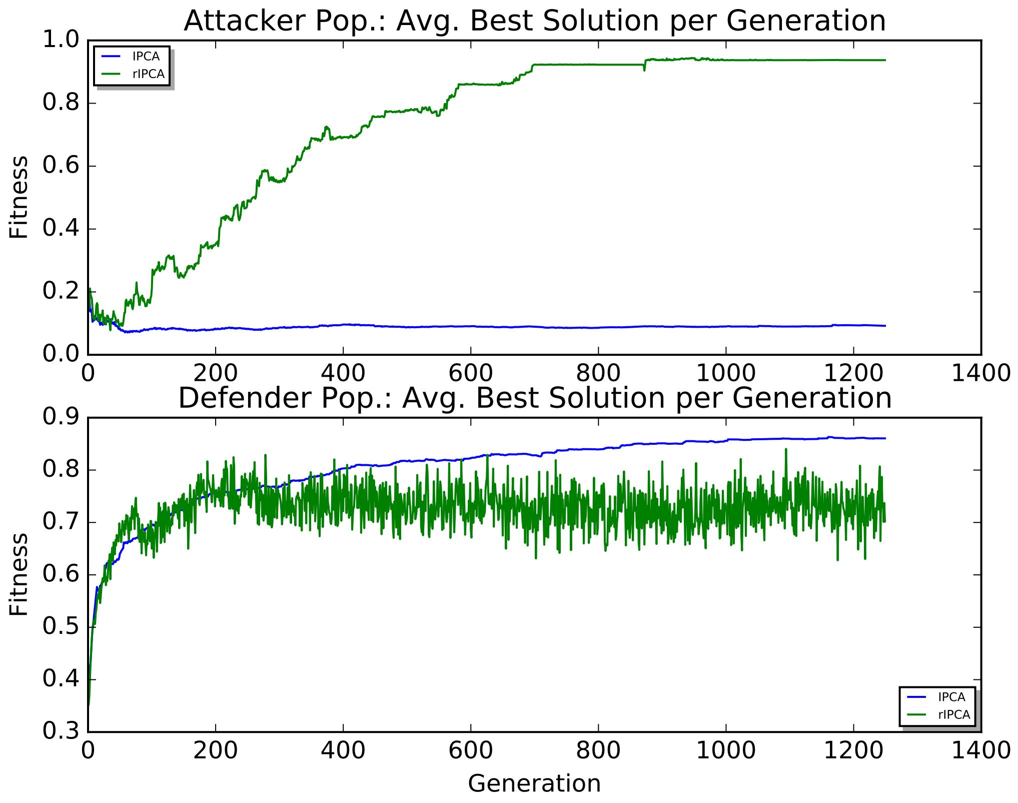


Figure 4-4: Best fitness value average (over 30 runs) per generation for 1250 generations on the Compare-on-one problem. Algorithms compared: IPCA and rIPCA

In Figure 4-5 we can see, over 1000 generations, how each algorithm performs. For this simulation, we compare the performance over the defending population as this is the population which we have set to optimize for. We note that IPCA shows monotonically increasing performance in its learner (defender) population, which follows our expectations. rIPCA follows IPCA in performance, however, its performance increases are not monotonic. Coev and MaxSolve perform similarly, however, the quality of their results is outmatched by both IPCA and rIPCA. MinMax notably performs poorly in this problem. This is due mostly to the nature of the problem and not MinMax itself.

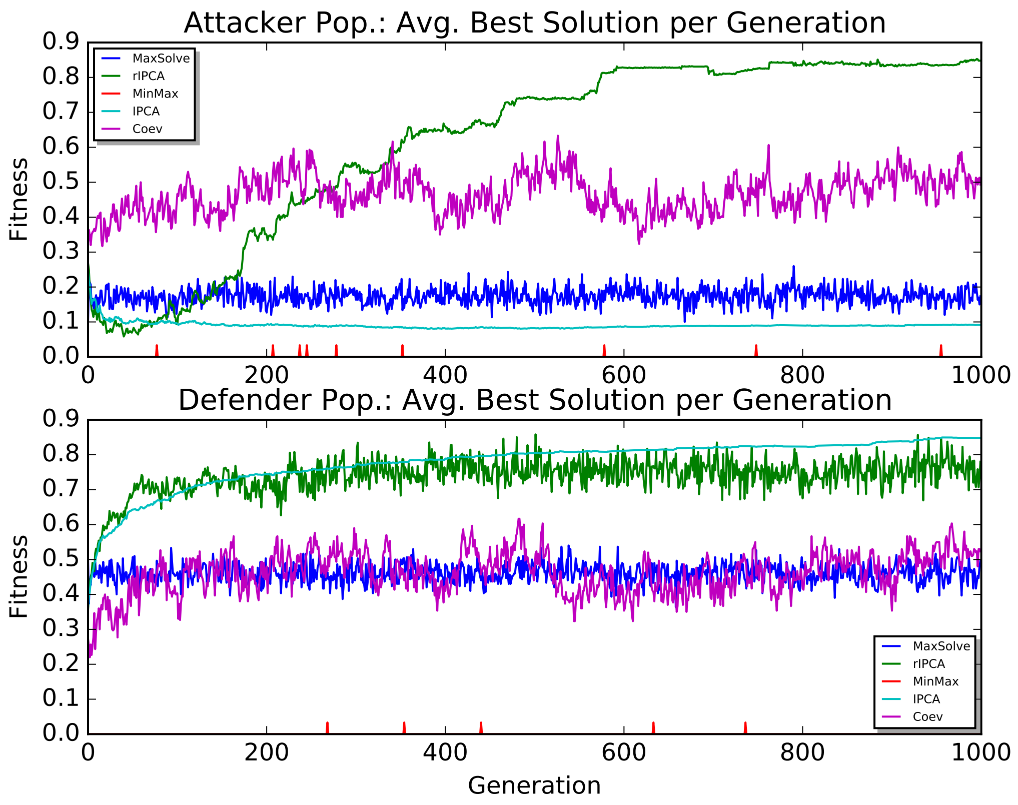


Figure 4-5: Best fitness value average (over 30 runs) per generation for 1000 generations on the Compare-on-one problem. Algorithms compared: IPCA, rIPCA, Coev, MinMax, MaxSolve.

Table 4.3: Network Mission results for coevolution over all topologies

	Topology 0		*Topology 1*		*Topology 2*	
Algorithm	**Exec Time(s)**	**Final Perf.**	**Exec Time(s)**	**Final Perf.**	**Exec Time(s)**	**Final Perf.**
Coev	10.417 ± 1.650	0.091 ± 0.014	36.911 ± 13.290	0.008 ± 0.001	180.114 ± 80.664	0.005 ± 0.000
MinMax	9.802 ± 1.693	0.045 ± 0.028	34.745 ± 10.351	0.005 ± 0.002	158.955 ± 72.101	0.004 ± 0.001
MaxSolve	20.945 ± 1.336	0.088 ± 0.022	12.322 ± 19.236	0.007 ± 0.001	768.817 ± 342.642	0.005 ± 0.001
IPCA	66.576 ± 6.537	0.097 ± 0.021	266.382 ± 59.253	0.008 ± 0.000	1729.165 ± 623.941	0.005 ± 0.000
rIPCA	47.754 ± 8.108	0.128 ± 0.055	267.784 ± 69.347	0.008 ± 0.000	1566.194 ± 643.867	0.005 ± 0.000

4.2.2 Network Missions

In Garcia et al., 2017, research published in conjunction with this thesis, we presented the results for network missions under an earlier version of our network simulator. Here we present our latest results using the most recent version of RIVALS in which chord, one of the defense protocols, now makes a distinction between the physical and logical layers of a network. This improves the quality of the results as the experiment more closely resembles a production environment.

As before with Compare-on-one, we run the network mission simulation over 30 runs and collect the average over the results. In Table 4.3 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 differentiate with IPCA and rIPCA being superior. We conjecture this is due to the test archives for both IPCA and rIPCA as these archives help enforce monotonic performance increases. When looking at Topologies 1 & 2, we do not notice 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 maintains to be on par or better than IPCA in both execution time and performance.

4.2.3 Strategic Network Resource Placement

Below we have included our timing and performance results for the strategic network resource placement problem. Using this information and the corresponding graphs, we can better understand how each of these algorithms performs under a simulated cyber security scenario.

Table 4.4: Network Resource Placement results for coevolution over all topologies

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

In Table 4.4 we see the averaged results over 30 runs for each configuration. The first three algorithms, Coev, MinMax, MaxSolve, perform quickly yet are out performed by both IPCA and rIPCA. Next, rIPCA, while not the best algorithm in terms of performance, is second place while consistently performing better than IPCA in terms execution time. This follows our previous results with regards to rIPCA's speed. Similarly, we see that rIPCA consistently performs better than Coev, MinMax and MaxSolve. It is important to note, however, that rIPCA's results vary more than IPCA and, as such, rIPCA's results are not seen to provide high consistency with respect to performance.

Next, in Figure 4-6, we show how each algorithm compares over time on Topology 0. Following with our previous results from Compare-on-one, IPCA continues to show monotonic increasing performance in the defending population. rIPCA's results are also consistent as the results show its performance as second only to IPCA's. MinMax is notably a poor performer while MaxSolve and Coev perform within the average of the group.

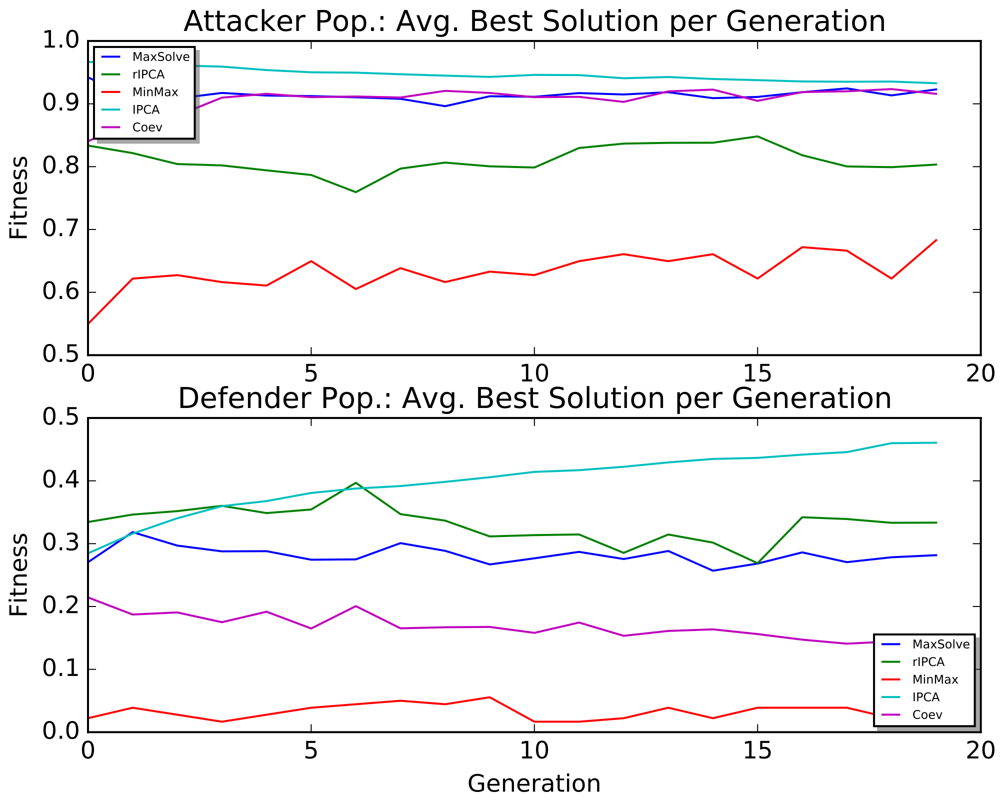


Figure 4-6: Best fitness value average (over 30 runs) per generation for 20 generations on the Strategic Network Resource Placement problem. Algorithms compared: IPCA, rIPCA, Coev, MinMax, MaxSolve.

Chapter 5

Discussion

This chapter reflects on the paths we took while conducting this research. In particular, we discuss the applicability of evolutionary algorithms in cybersecurity. Moreover, once having considered their usefulness we explore the trade-offs one makes when using this variety of algorithms.

5.1 Applicability of Evolutionary Algorithms

In this section we elaborate on the use of evolutionary algorithms with respect to cybersecurity. Currently, problems of this nature are commonly dealt with manually. While manual exploration of these issues can be lead to good results through experience and professional knowledge, it is also not suitable for all cases and often impossible to do on the scale of large networks. As such, evolutionary algorithms provide a strong alternative to these issues. However, evolutionary algorithms are also limited in that, depending on the problem, their simulation may run slow or even produce weak results. Thus, it is important that their implementations be properly calibrated and tested before they are used on larger problems.

Our plan with this project was first to calibrate a suite of algorithms against a standard coevolutionary benchmark problem, Compare-on-one, so that we could tweak our suite for speed and performance. This was in preparation for our integration with the network simulator used by RIVALS. Integration with the network simulator proved to be successful. Although, as the network simulator grew to be more complex, the execution times for our simulations started to become simulation heavy rather than algorithm heavy. This proved our assumption that for us to be able to apply evolutionary algorithms to cybersecurity, our algorithms needed to be able to improve on execution time while still performing well. By minimizing simulation evaluations in rIPCA, we were able to provide a system that could perform well, yet also perform quickly.

5.2 Speed vs. Quality of Results

In this section we discuss how we manage both speed of execution and quality of results for our system. As seen before in Chapter 3, our suite of algorithms contain algorithms with strong guarantees with respect to the quality of there results (i.e. IPCA & monotonic-increasing performance of learner individuals) as well as algorithms which are fast yet produce weak results. For us, we have to assume that our model is complex and thus performing any evaluation of fitness of an individual should be assumed to be expensive. As such, when we optimized the speed of an algorithm we worked to minimize the amount of expected fitness evaluations rather than focus necessarily on its wall-clock execution time, though these were often correlated.

In IPCA, we note that the weak learner individuals are pruned through the process of non-dominated filtering while test individuals are kept for the duration of the experiment. This configuration allows IPCA to guarantee monotonic-increasing fitness over an experiment. We must note, however, that fitness evaluations are performed population vs. population. As such, the minimum number of evaluations will be the product of the population sizes. Under IPCA, this implies that as generations pass, more and more evaluations will be necessary since the test population is allowed to grow indefinitely. Given our assumption of expensive fitness calls, this made IPCA impractical for experiments with a high number of generations to run. Our solution was then to apply concepts already in IPCA to minimize the amount of fitness evaluations. This proved to be useful even without the guarantees provided by IPCA.

Chapter 6

Conclusions & Future Work

Through this work, we have analyzed a suite of coevolutionary algorithms on a variety of settings and have introduced our own variation of an existing algorithm. Our variation, rIPCA, was shown to be on par or close to IPCA in terms of performance (quality of results) while surpassing IPCA, sometimes by large margins, in terms of execution speed. We also showed that we were successful in applying them not only to Compare-on-one, a standard Coevolutionary benchmark problem, but also to our own network simulator, RIVALS. With these capabilities, we are able to simulate the evolution of an attacker with DOS capabilities over any given network topology.

In our future work, we will continue to improve the performance and speed of the coevolutionary algorithms as well as try algorithms with different solution concepts, e.g. Nash Equilibrium (Ficici and Pollack, 2003). Additional algorithms we look forward to experimenting with are DISCO (Liskowski and Krawiec, 2014) and DOF (Liskowski and Krawiec, 2016). Both algorithms work to approximate fitness values in hopes of decreasing the time necessary to calculate them.

Bibliography

Dalvi, Nilesh et al. (2004). “Adversarial classification”. In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, pp. 99–108.

De Jong, Edwin (2005). “The maxsolve algorithm for coevolution”. In: *Proceedings of the 7th annual conference on Genetic and evolutionary computation.* ACM, pp. 483–489.

Ficici, Sevan G and Jordan B Pollack (2003). “A game-theoretic memory mechanism for coevolution”. In: *Genetic and Evolutionary Computation Conference.* Springer, pp. 286–297.

Garcia, D. et al. (2017). “Investigating Coevolutionary Archive Based Genetic Algorithms on Cyber Defense Networks”. In: *Proceedings of the 19th Annual Conference on Genetic and Evolutionary Computation.* GECCO ’17. ACM. DOI: `10.475/1234`.

Haddadi, Fariba 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, pp. 893–900.

Harper, Robin (2014). “Evolving robocode tanks for Evo robocode”. In: *Genetic Programming and Evolvable Machines* 15.4, pp. 403–431.

Hemberg, Erik et al. (2016). “Detecting tax evasion: a co-evolutionary approach”. In: *Artificial Intelligence and Law* 24.2, pp. 149–182.

Hingston, P. and M. Preuss (2011). “Red teaming with coevolution”. In: *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pp. 1155–1163. DOI: `10.1109/CEC.2011.5949747`.

Jong, Edwin D de (2007). “A Monotonic Archive for Pareto-Coevolution”. In: *Evolutionary Computation* 15.1, pp. 61–93.

Krawiec, Krzysztof and Malcolm Heywood (2016). “Solving Complex Problems with Coevolutionary Algorithms”. In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion.* ACM, pp. 687–713.

Liskowski, Paweł and Krzysztof Krawiec (2014). “Discovery of implicit objectives by compression of interaction matrix in test-based problems”. In: *International Conference on Parallel Problem Solving from Nature.* Springer, pp. 611–620.

— (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, pp. 749–756.

O'Neill, Michael and Conor Ryan (2003). *Grammatical evolution: evolutionary automatic programming in an arbitrary language*. Vol. 4. Springer.

Popovici, Elena et al. (2012). "Coevolutionary principles". In: *Handbook of Natural Computing*. Springer, pp. 987–1033.

Rush, George, 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, pp. 859–866.

Service, Travis and Daniel Tauritz (2009). "Increasing infrastructure resilience through competitive coevolution". In: *New Mathematics and Natural Computation* 5.02, pp. 441–457.

Silva Arantes, Jesimar da et al. (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*, pp. 486–493. DOI: `10.1109/ICTAI.2015.78`. URL: `http://dx.doi.org/10.1109/ICTAI.2015.78`.

Stoica, Ion et al. (2001). "Chord: A scalable peer-to-peer lookup service for internet applications". In: *ACM SIGCOMM Computer Communication Review* 31.4, pp. 149–160.

Wagner, Stuart et al. (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, pp. 39–46.

Whigham, Peter A et al. (2015). "Examining the Best of Both Worlds of Grammatical Evolution". In: *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*. ACM, pp. 1111–1118.

Winterrose, Michael L 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, p. 9.