Evolutionary Computing Approaches for Wickedly Hard National Security Problems

Daniel Tauritz, PhD

Director, Biomimetic National Security Artificial Intelligence (BONSAI) Laboratory
Academic Director, Cyber Security Sciences Institute
Chief Cyber AI Strategist, Auburn Cyber Research Center
Associate Professor, Department of Computer Science and Software Engineering
Auburn University

November 18, 2019

Part I: Preliminaries

Terminology

- Many computational problems can be formulated as **generate-and-test** search problems

Terminology

- Many computational problems can be formulated as **generate-and-test** search problems
- A **search space** contains the set of all possible solutions

Terminology

- Many computational problems can be formulated as **generate-and-test** search problems
- A **search space** contains the set of all possible solutions
- A **search space generator** is *complete* if it can generate the entire search space

Terminology

- Many computational problems can be formulated as **generate-and-test** search problems
- A **search space** contains the set of all possible solutions
- A **search space generator** is *complete* if it can generate the entire search space
- An **objective function** tests the quality of a solution

Terminology

- Many computational problems can be formulated as **generate-and-test** search problems
- A **search space** contains the set of all possible solutions
- A **search space generator** is *complete* if it can generate the entire search space
- An **objective function** tests the quality of a solution
- A **heuristic** is a problem-dependent rule-of-thumb

Terminology

- Many computational problems can be formulated as **generate-and-test** search problems
- A **search space** contains the set of all possible solutions
- A **search space generator** is *complete* if it can generate the entire search space
- An **objective function** tests the quality of a solution
- A **heuristic** is a problem-dependent rule-of-thumb
- A **meta-heuristic** determines the sampling order over a search space with the goal to find a near-optimal solution (or set of solutions)

Terminology

- Many computational problems can be formulated as **generate-and-test** search problems
- A **search space** contains the set of all possible solutions
- A **search space generator** is *complete* if it can generate the entire search space
- An **objective function** tests the quality of a solution
- A **heuristic** is a problem-dependent rule-of-thumb
- A **meta-heuristic** determines the sampling order over a search space with the goal to find a near-optimal solution (or set of solutions)
- A **hyper-heuristic** is a meta-heuristic for a space of programs

Algorithmic Toolbox

- A **Black-Box Search Algorithm (BBSA)** is a meta-heuristic which iteratively generates trial solutions employing solely the information gained from previous trial solutions, but no explicit problem knowledge

Algorithmic Toolbox

- A **Black-Box Search Algorithm (BBSA)** is a meta-heuristic which iteratively generates trial solutions employing solely the information gained from previous trial solutions, but no explicit problem knowledge
- **Evolutionary Algorithms (EAs)** can be described as a class of *stochastic*, *population-based* BBSAs inspired by *Evolution Theory*, *Genetics*, and *Population Dynamics*

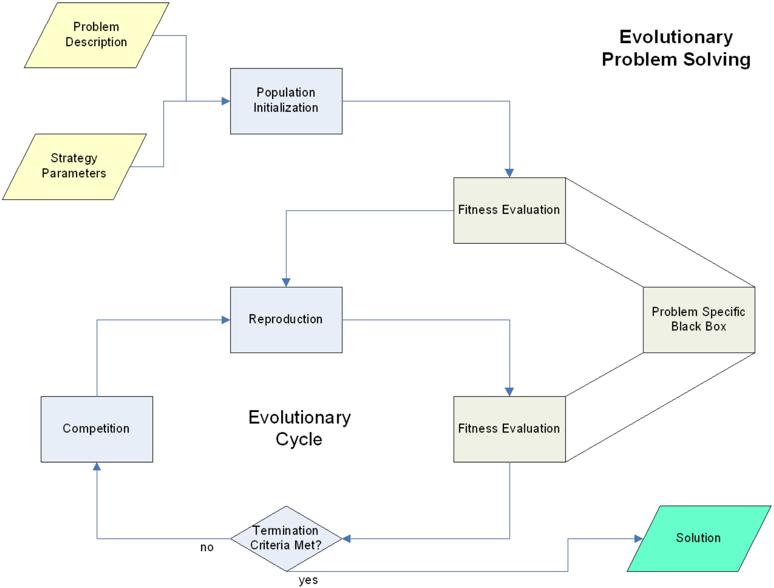
Algorithmic Toolbox

- A **Black-Box Search Algorithm (BBSA)** is a meta-heuristic which iteratively generates trial solutions employing solely the information gained from previous trial solutions, but no explicit problem knowledge
- **Evolutionary Algorithms (EAs)** can be described as a class of *stochastic*, *population-based* BBSAs inspired by *Evolution Theory*, *Genetics*, and *Population Dynamics*
- **Genetic Programming (GP)** is a type of EA for searching a space of programs

Algorithmic Toolbox

- A **Black-Box Search Algorithm (BBSA)** is a meta-heuristic which iteratively generates trial solutions employing solely the information gained from previous trial solutions, but no explicit problem knowledge
- **Evolutionary Algorithms (EAs)** can be described as a class of *stochastic*, *population-based* BBSAs inspired by *Evolution Theory*, *Genetics*, and *Population Dynamics*
- **Genetic Programming (GP)** is a type of EA for searching a space of programs
- Hyper-heuristics are predominantly implemented with GP; they work by trading off adequate performance on general problem classes with high performance on targeted problem classes

Evolutionary Cycle



Genetic Programming

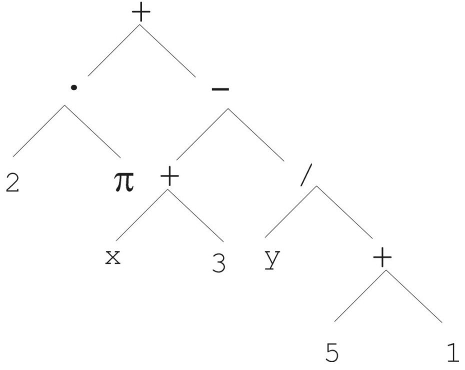
- EA with Hierarchical Representation for Model Identification

Genetic Programming

- EA with Hierarchical Representation for Model Identification
- Koza style Tree GP is the most prevalent

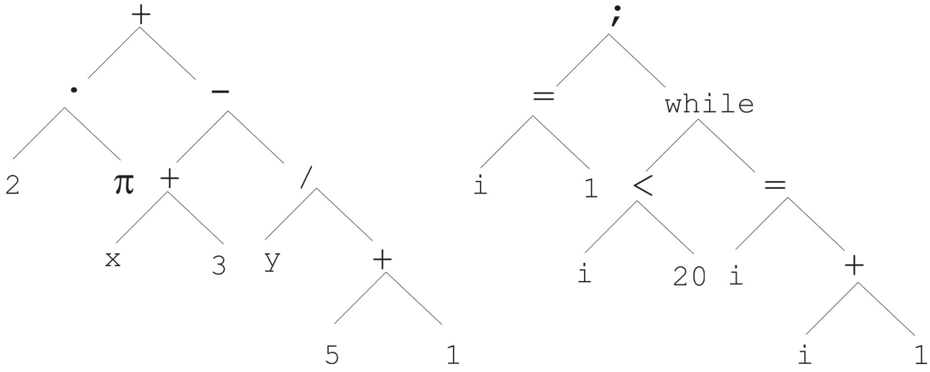
Genetic Programming

- EA with Hierarchical Representation for Model Identification
- Koza style Tree GP is the most prevalent

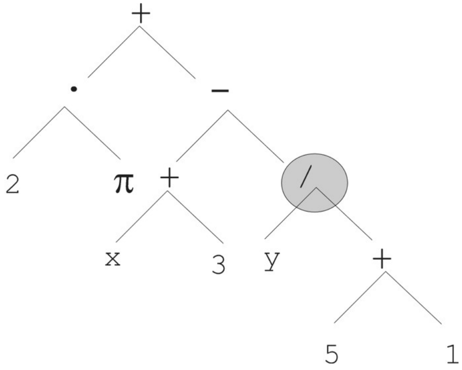


Genetic Programming

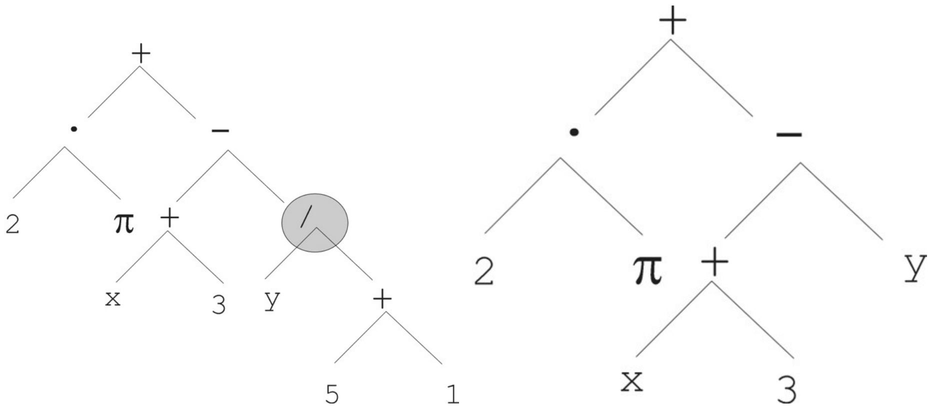
- EA with Hierarchical Representation for Model Identification
- Koza style Tree GP is the most prevalent



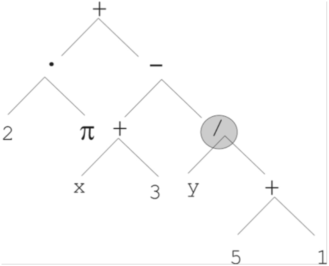
Genetic Programming - Mutation



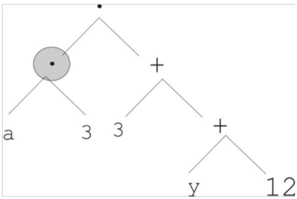
Genetic Programming - Mutation



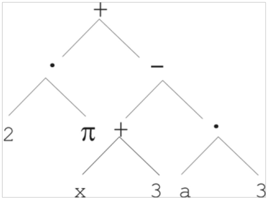
Genetic Programming - Recombination



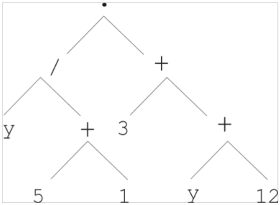
Parent 1



Parent 2



Child 1



Child 2

Part II: Hyper-heuristics for National Security Problems

Automated Design of Network Security Metrics

Aaron Pope[1,2], Robert Morning[2], Daniel Tauritz[1,2], Alexander Kent[2]

2018 Genetic and Evolutionary Computation Conference (GECCO 2018)

Kyoto, Japan, July 15-19, 2018

[1]Missouri University of Science and Technology

[2]Los Alamos National Laboratory

Motivation

- Cybersecurity is a rapidly changing environment

Motivation

- Cybersecurity is a rapidly changing environment
- "Best practices" are not enough to protect a network

Motivation

- Cybersecurity is a rapidly changing environment
- "Best practices" are not enough to protect a network
- Administrators rely on tools to assess security for large networks

Motivation

- Cybersecurity is a rapidly changing environment
- "Best practices" are not enough to protect a network
- Administrators rely on tools to assess security for large networks
- Analysis methods need to be created for new attack techniques

Motivation

- Cybersecurity is a rapidly changing environment
- "Best practices" are not enough to protect a network
- Administrators rely on tools to assess security for large networks
- Analysis methods need to be created for new attack techniques
- Solutions can be complex and unintuitive

Motivation

- Cybersecurity is a rapidly changing environment
- "Best practices" are not enough to protect a network
- Administrators rely on tools to assess security for large networks
- Analysis methods need to be created for new attack techniques
- Solutions can be complex and unintuitive
- Manual development is too slow for rapid response

Problem

Given an attack model, is it possible to automatically generate a useful security metric?

Problem

Given an attack model, is it possible to automatically generate a useful security metric?

- Model the attack using graph representation of network

Problem

Given an attack model, is it possible to automatically generate a useful security metric?

- Model the attack using graph representation of network
- Use hyper-heuristic techniques to generate novel graph-based security metrics

Problem

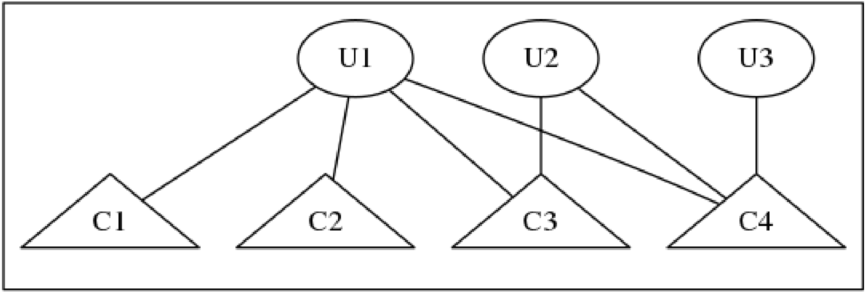
Given an attack model, is it possible to automatically generate a useful security metric?

- Model the attack using graph representation of network
- Use hyper-heuristic techniques to generate novel graph-based security metrics
- Evaluate metrics by how well they predict attack success

Network Representation

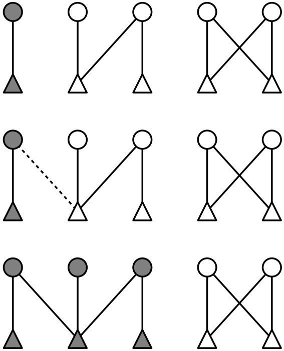
Bipartite Authentication Graphs (BAG)

- Vertices represent networked hosts and user accounts
- Edges indicate an account being used to access a host
- Especially useful for centralized single-sign-on systems



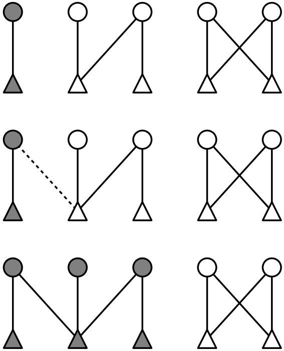
Credential Theft Attack Model

1. Adversary starts on a random host
2. Harvests credentials to follow legitimate user traffic
3. Repeats, traversing a growing portion of the network



Credential Theft Attack Model

1. Adversary starts on a random host
2. Harvests credentials to follow legitimate user traffic
3. Repeats, traversing a growing portion of the network



Given a specific credential policy and time limit, what is the expected portion of the network an adversary can reach?

Credential Policies

Possible credential policies:

- Time limit on credential expiration

Credential Policies

Possible credential policies:

- Time limit on credential expiration
- Maximum number of credentials stored

Credential Policies

Possible credential policies:

- Time limit on credential expiration
- Maximum number of credentials stored
- Periodic clearing of credential cache

Credential Policies

Possible credential policies:

- Time limit on credential expiration
- Maximum number of credentials stored
- Periodic clearing of credential cache
- Some combination of these or others

Compact Network Representation

- Dynamic graph representations for entire days are too complex

Compact Network Representation

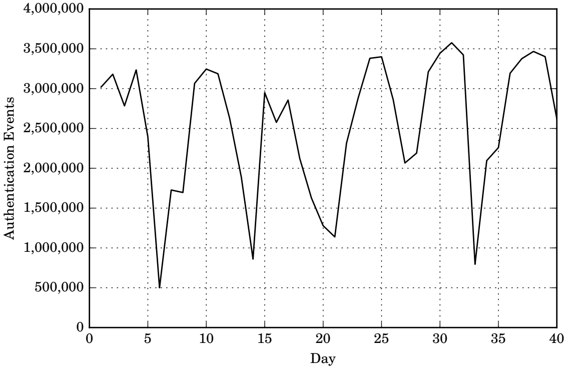
- Dynamic graph representations for entire days are too complex
- Simpler, static input graphs are produced for each day

Compact Network Representation

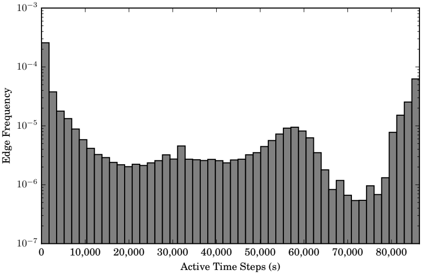
- Dynamic graph representations for entire days are too complex
- Simpler, static input graphs are produced for each day
- Edge weights equal the number of time-steps an authentication edge is active that day

LANL Authentication Dataset Details

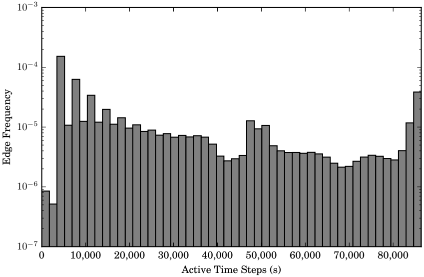
Unique Users	10,044
Unique Computers	15,779
Unique (User, Computer) Pairs	124,020
Total Authentication Events	101,918,344
Average Daily Authentication Events	2,547,959



Simulation Policies Considered



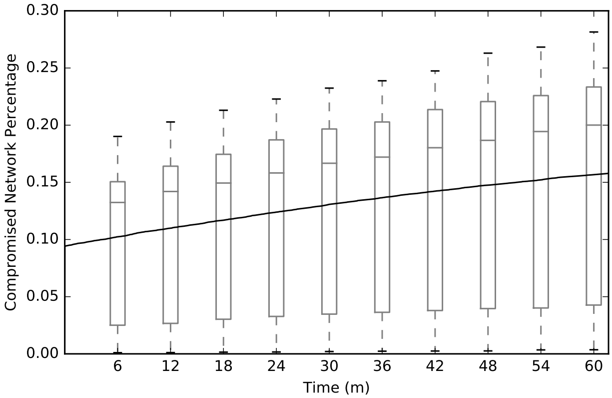
10 credential limit policy



1 hour expiration policy

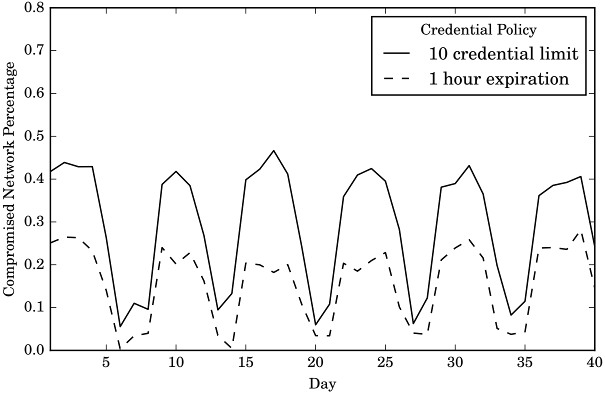
Distribution of authentication edge activity levels

Simulation Results



1-hour ticket lifetime policy simulation results

Simulation Results



Daily simulation results for both credential policies

Hyper-Heuristic Approach

- Use strongly-typed genetic programming to evolve an algorithm to predict attack success

Hyper-Heuristic Approach

- Use strongly-typed genetic programming to evolve an algorithm to predict attack success
- Solutions take a graph representation of the network as input and return a predicted percentage

Hyper-Heuristic Approach

- Use strongly-typed genetic programming to evolve an algorithm to predict attack success
- Solutions take a graph representation of the network as input and return a predicted percentage
- Measure evolved solution's output against simulation results

$$fitness = -\frac{\sum_{d \in days} \left| \frac{simulated\_result - predicted\_result}{simulated\_result} \right|}{|days|}$$

GP Primitive Set

Inspired by previous work evolving graph-based heuristics

- Math operations: add, subtract, etc.
- Numerical constants: integer or probability
- Boolean nodes: true, false, random
- Control flow: if [else], for, while
- Graph elements: vertices, edges
- Local and global graph metrics: average degree, centrality measures
- Collection manipulation: concatenation, filtering, mapping
- Subgraph induction

GP Parameter Values

Parameter	Value[3]
Population size	400
Offspring per generation	600
Parent selection tournament size	8
Minimum initial parse tree height	4
Maximum initial parse tree height	7
Recombination probability	70%
Mutation probability	30%
Convergence threshold	10

[3] Values tuned by a random-restart hill-climbing search.

Evolved Solutions

- Complex (200+ lines of code)
- Common functional elements:
  1. Induce a subgraph with the most active edges
  2. Find the connected components in the induced graph
  3. Filter out the account vertices in each component vertex set
  4. Return a value based on the number of computers in each component relative to the number of computers in the original graph

Comparison of Evolved Metric Heuristics

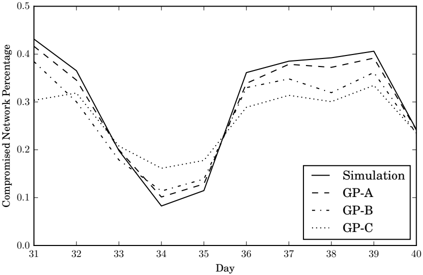
	10 Credential Limit		**1 Hour Expiration**	
Method	**Result**	**Error**	**Result**	**Error**
Simulation	29.797%	N/A	17.484%	N/A
GP-A	29.151%	6.15%	21.411%	60.93%
GP-B	27.093%	14.85%	17.571%	11.23%
GP-C	26.427%	28.00%	20.387%	71.79%

GP-A: heuristic trained on 10 credential limit policy

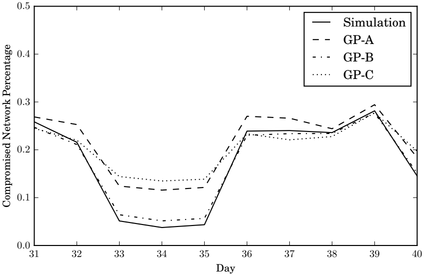
GP-B: heuristic trained on 1 hour expiration policy

GP-C: heuristic trained on combined data from both policies

Comparison of Evolved Metric Heuristics



10 credential limit policy



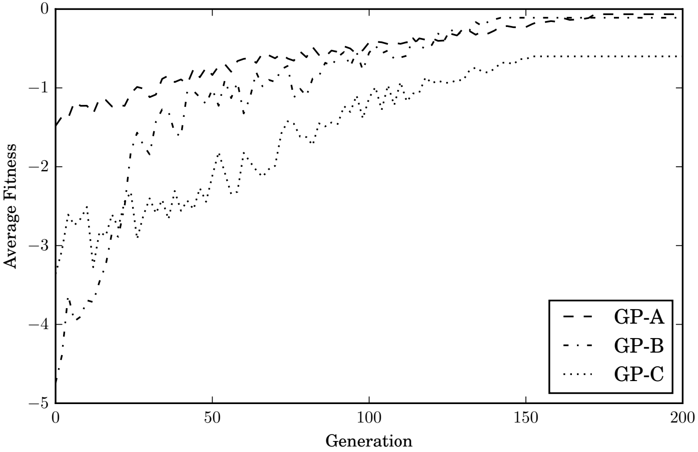
1 hour expiration policy

GP-A: heuristic trained on 10 credential limit policy

GP-B: heuristic trained on 1 hour expiration policy

GP-C: heuristic trained on combined data from both policies

Evolution Progress



Take-Away

- Automated design of security metrics would be helpful in assessing vulnerabilities to new attacks even before they are fully understood.

Take-Away

- Automated design of security metrics would be helpful in assessing vulnerabilities to new attacks even before they are fully understood.
- In this work, the hyper-heuristic was trained against a simulation, but this could be replaced with penetration testing or real-world incident data.

Take-Away

- Automated design of security metrics would be helpful in assessing vulnerabilities to new attacks even before they are fully understood.
- In this work, the hyper-heuristic was trained against a simulation, but this could be replaced with penetration testing or real-world incident data.
- This work demonstrates the potential of hyper-heuristics in finding novel network security metrics with less reliance on subject matter expertise.

Scalable Automated Tailoring of SAT Solvers with Cyber Security Applications

Funded by Sandia National Laboratories

Sandia collaborators are Samuel Mulder, Denis Bueno, Shelly Leger, Richard Barrett, and Alex Bertels

Boolean Satisfiability Problem (SAT)

- A SAT instance is a boolean formula, typically in Conjunctive Normal Form (CNF) like:

Example SAT Instance in 3-CNF

$$(x_1 \lor x_3 \lor \neg x_2) \land (x_2 \lor \neg x_1 \lor \neg x_3) \land (\neg x_2 \lor x_3 \lor x_1)$$

Boolean Satisfiability Problem (SAT)

- A SAT instance is a boolean formula, typically in Conjunctive Normal Form (CNF) like:

Example SAT Instance in 3-CNF

$$(x_1 \vee x_3 \vee \neg x_2) \wedge (x_2 \vee \neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee x_1)$$

- Solving a SAT instance means deciding whether there exists an assignment of truth values for its boolean variables to make the formula true (i.e., satisfy the instance)

Boolean Satisfiability Problem (SAT)

- A SAT instance is a boolean formula, typically in Conjunctive Normal Form (CNF) like:

Example SAT Instance in 3-CNF

$$(x_1 \vee x_3 \vee \neg x_2) \wedge (x_2 \vee \neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee x_1)$$

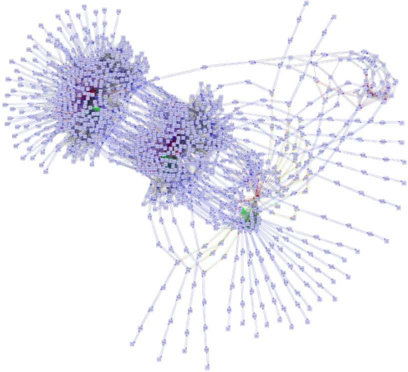
- Solving a SAT instance means deciding whether there exists an assignment of truth values for its boolean variables to make the formula true (i.e., satisfy the instance)
- NP Complete

SAT Applications

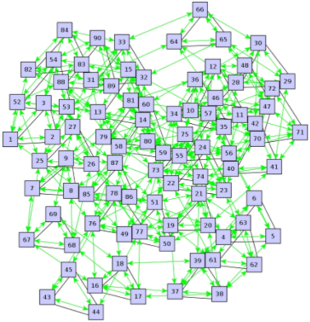
- Encryption at Galois Inc.
- Embedded circuits of Centaur Technology
- Repairing cosmic ray damage of FPGAs at NASA
- Designing the Intel Core i7 processors
- Mapping out mutations in DNA
- Static code analysis
- Program understanding for cyber security

SAT Instance Structure

Long Multiplier



Graph Coloring



Evolving CDCL Heuristics

- For each problem class mapped to SAT, there is a structure optimal SAT solver

Evolving CDCL Heuristics

- For each problem class mapped to SAT, there is a structure optimal SAT solver
- Among the most efficient known SAT solvers are conflict-driven clause learning (CDCL) solvers

Evolving CDCL Heuristics

- For each problem class mapped to SAT, there is a structure optimal SAT solver
- Among the most efficient known SAT solvers are conflict-driven clause learning (CDCL) solvers
- The *Automated Design of Boolean Satisfiability Problem Solvers Employing Evolutionary Computing (ADSSEC) system* evolves CDCL SAT solvers to target arbitrary, but particular, SAT classes

ADSSEC

- Is a generative hyper-heuristic framework

ADSSEC

- Is a generative hyper-heuristic framework
- Employs Koza-style Genetic Programming (GP) trees

ADSSEC

- Is a generative hyper-heuristic framework
- Employs Koza-style Genetic Programming (GP) trees
- As a first step, automates the design of new variable scoring heuristics

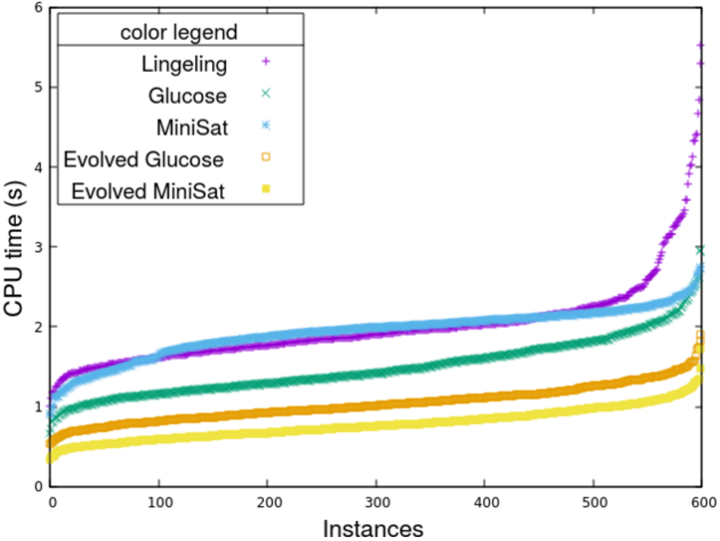
ADSSEC

- Is a generative hyper-heuristic framework
- Employs Koza-style Genetic Programming (GP) trees
- As a first step, automates the design of new variable scoring heuristics
- Evaluates newly generated heuristics by replacing the defaults in the Minisat and Glucose SAT solvers

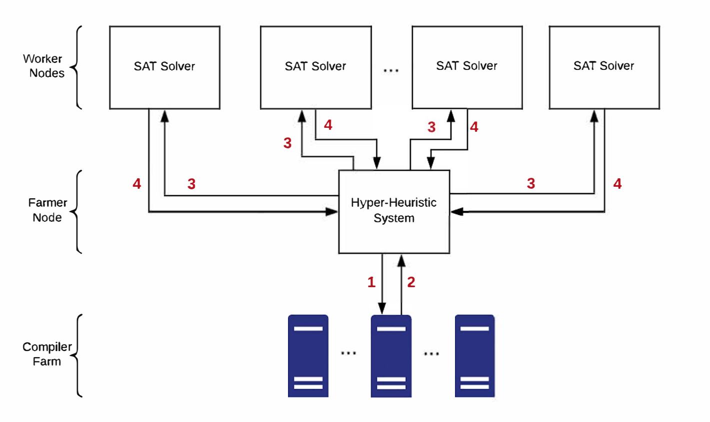
ADSSEC

- Is a generative hyper-heuristic framework
- Employs Koza-style Genetic Programming (GP) trees
- As a first step, automates the design of new variable scoring heuristics
- Evaluates newly generated heuristics by replacing the defaults in the Minisat and Glucose SAT solvers
- Employs a novel asynchronous parallel evolutionary algorithm (APEA)

Comparison to a state-of-the-art solver on unif-k5 dataset



MPI Cluster Parallelization Approach



Grand Challenges in Hyper-heuristics

- Algorithmic Primitive Granularity Control

Grand Challenges in Hyper-heuristics

- Algorithmic Primitive Granularity Control
- Automated Decomposition & Recomposition of Algorithmic Primitives

Grand Challenges in Hyper-heuristics

- Algorithmic Primitive Granularity Control
- Automated Decomposition & Recomposition of Algorithmic Primitives
- Automated Extraction of Algorithmic Primitives

Part III: Coevolution for Adversarial National Security Problems

Real-World Game-Theoretic Problems

- Game Theory: multi-agent problem with conflicting utility functions

Real-World Game-Theoretic Problems

- Game Theory: multi-agent problem with conflicting utility functions
- Real-world examples:
  - economic & military strategy

Real-World Game-Theoretic Problems

- Game Theory: multi-agent problem with conflicting utility functions
- Real-world examples:
  - economic & military strategy
  - arms control

Real-World Game-Theoretic Problems

- Game Theory: multi-agent problem with conflicting utility functions
- Real-world examples:
  - economic & military strategy
  - arms control
  - auctions

Real-World Game-Theoretic Problems

- Game Theory: multi-agent problem with conflicting utility functions
- Real-world examples:
  - economic & military strategy
  - arms control
  - auctions
  - cyber security

Real-World Game-Theoretic Problems

- Game Theory: multi-agent problem with conflicting utility functions
- Real-world examples:
  - economic & military strategy
  - arms control
  - auctions
  - cyber security
- Common problem: real-world games are typically incomputable

Real-World Game-Theoretic Problems

- Game Theory: multi-agent problem with conflicting utility functions
- Real-world examples:
  - economic & military strategy
  - arms control
  - auctions
  - cyber security
- Common problem: real-world games are typically incomputable
- Solution: Computational Game Theory

Approximating Incomputable Games

- Consider the space of each user's actions

Approximating Incomputable Games

- Consider the space of each user's actions
- Perform local search in these spaces

Approximating Incomputable Games

- Consider the space of each user's actions
- Perform local search in these spaces
- Solution quality in one space is dependent on the search in the other spaces

Approximating Incomputable Games

- Consider the space of each user's actions
- Perform local search in these spaces
- Solution quality in one space is dependent on the search in the other spaces
- The simultaneous search of co-dependent spaces is naturally modeled as an armsrace

Classical Computational Solver Limitations

Complex real-world problems can be (practically) unsolvable with classic approaches

- Black box

Classical Computational Solver Limitations

Complex real-world problems can be (practically) unsolvable with classic approaches

- Black box
- “Ill-behaved” search space

Classical Computational Solver Limitations

Complex real-world problems can be (practically) unsolvable with classic approaches

- Black box
- “Ill-behaved” search space
- Intractable

Classical Computational Solver Limitations

Complex real-world problems can be (practically) unsolvable with classic approaches

- Black box
- “Ill-behaved” search space
- Intractable
- Evolution has a demonstrated ability to solve very complex problems

Coevolutionary Algorithm (CoEA)

CoEAs are a special type of EAs where the fitness of an individual is dependent on other individuals (i.e., individuals are explicitly part of the environment)

Coevolutionary Algorithm (CoEA)

CoEAs are a special type of EAs where the fitness of an individual is dependent on other individuals (i.e., individuals are explicitly part of the environment)

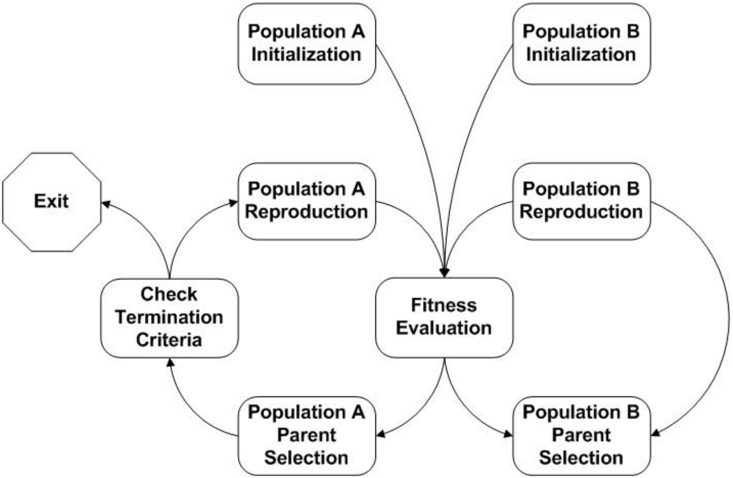
- Single species vs. multiple species

Coevolutionary Algorithm (CoEA)

CoEAs are a special type of EAs where the fitness of an individual is dependent on other individuals (i.e., individuals are explicitly part of the environment)

- Single species vs. multiple species
- Cooperative vs. competitive coevolution

Two-Population Competitive Coevolutionary Cycle



Coevolutionary Cyber Security

- Cyber Security is naturally modeled as a competitive multi-population coevolution (1 defender population and *n* adversary populations)

Coevolutionary Cyber Security

- Cyber Security is naturally modeled as a competitive multi-population coevolution (1 defender population and *n* adversary populations)
- Builds on previous NC-LAB experience coevolving attacks and defenses for electric transmission grid protection

Coevolving Attacker and Defender Strategies for Large Infrastructure Networks (CEADS-LIN)

Funded by Los Alamos National Laboratory (LANL) via the LANL/S&T Cyber Security Sciences Institute (CSSI)

Many moving parts

- Multi-Agent System

Many moving parts

- Multi-Agent System
- Representations

Many moving parts

- Multi-Agent System
- Representations
- Genetic Programming

Many moving parts

- Multi-Agent System
- Representations
- Genetic Programming
- Attack Models

Many moving parts

- Multi-Agent System
- Representations
- Genetic Programming
- Attack Models
- Fitness Functions

Many moving parts

- Multi-Agent System
- Representations
- Genetic Programming
- Attack Models
- Fitness Functions
- Simulation versus Emulation

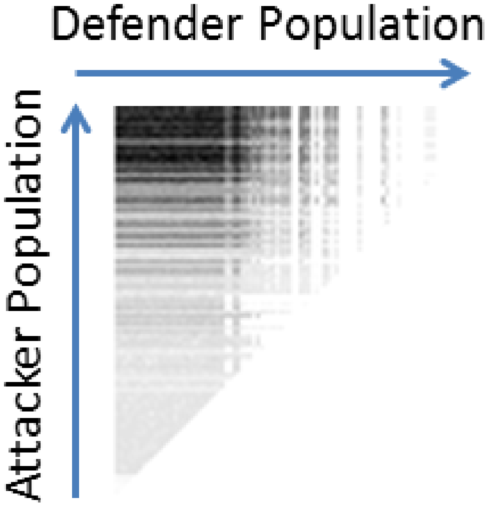
Many moving parts

- Multi-Agent System
- Representations
- Genetic Programming
- Attack Models
- Fitness Functions
- Simulation versus Emulation
- Hardware-based Emulation: EmuLab, DeterLab

Many moving parts

- Multi-Agent System
- Representations
- Genetic Programming
- Attack Models
- Fitness Functions
- Simulation versus Emulation
- Hardware-based Emulation: EmuLab, DeterLab
- Virtualized Network Emulation

CIAO Plot Example



Experimental CIAO Plots

TABLE II
TYPICAL RESULTING CIAO PLOTS (ATTACKER PERSPECTIVE)

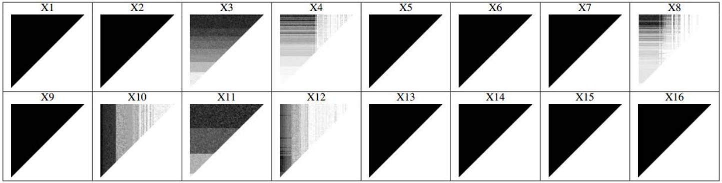
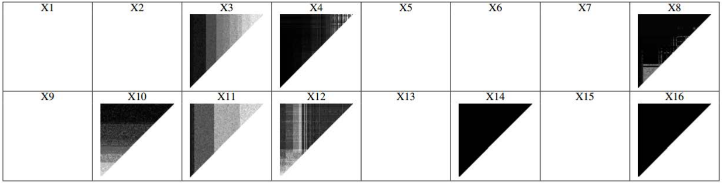
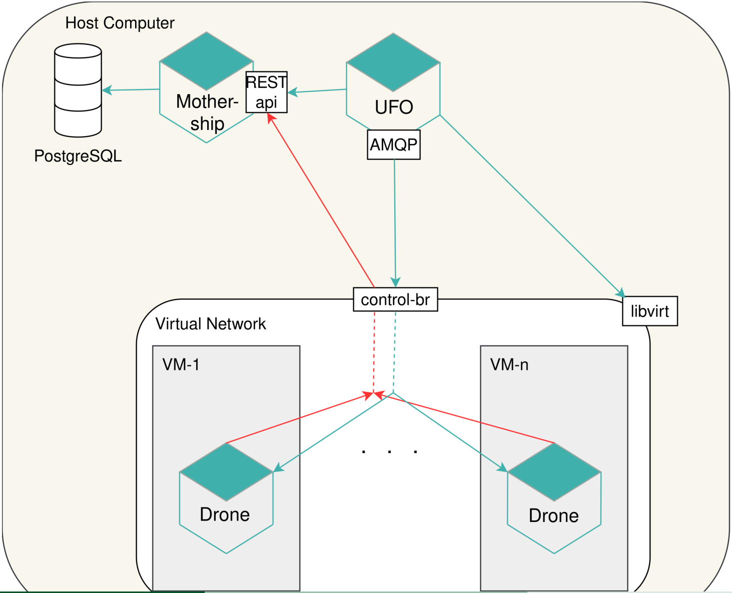


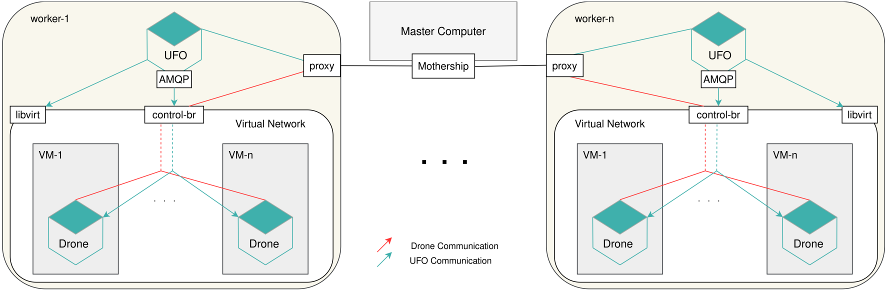
TABLE III
TYPICAL RESULTING CIAO PLOTS (DEFENDER PERSPECTIVE)



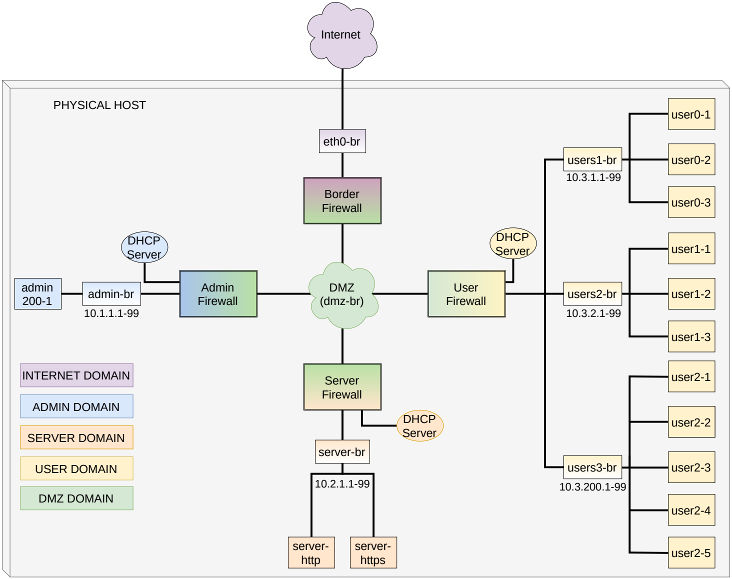
CEADS Architecture



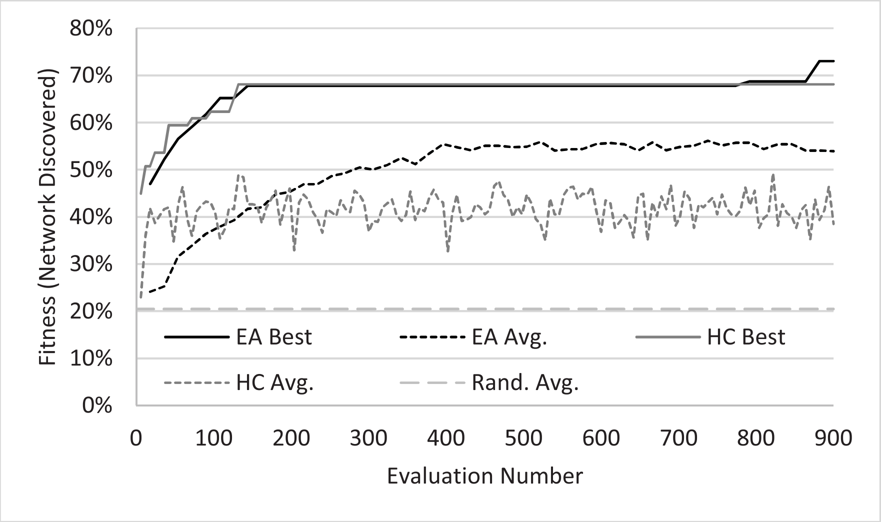
Distributed Architecture



Network Layout



Results


80%
70%
60%
50%
40%
30%
20%
10%
0%
Fitness (Network Discovered)
0
100
200
300
400
500
600
700
800
900
Evaluation Number
EA Best
EA Avg.
HC Best
HC Avg.
Rand. Avg.

Grand Challenges in Coevolving Attacker & Defender Strategies for Large Computer Networks

- Time Dilation in Computer Network Emulation

Grand Challenges in Coevolving Attacker & Defender Strategies for Large Computer Networks

- Time Dilation in Computer Network Emulation
- Simulating Human Users

Grand Challenges in Coevolving Attacker & Defender Strategies for Large Computer Networks

- Time Dilation in Computer Network Emulation
- Simulating Human Users
- Scaling & Operationalizing the CEADS-LIN System

Part IV: Take Home Message

Many wickedly hard national security problems map to computational problems that require...

Many wickedly hard national security problems map to computational problems that require...

- repeated solving of instances of the same problem class, can be effectively addressed with genetic programming based hyper-heuristics

Many wickedly hard national security problems map to computational problems that require...

- repeated solving of instances of the same problem class, can be effectively addressed with genetic programming based hyper-heuristics
- solving game theoretic problem instances, can be effectively addressed with coevolutionary algorithms applied to high-fidelity emulations