SAND2011-6035P

Adaptive Rule-Based Malware Detection Employing Learning Classifier Systems

Master's Thesis Defense

Missouri University of Science and Technology
Department of Computer Science
Jonathan Blount

Committee:
Dr. Daniel Tauritz (Advisor)
Dr. Bruce McMillin
Dr. Samuel Mulder (Sandia National Laboratories)

August 4, 2011

Outline

Motivation

- Cyber threats are quickly increasing in frequency and impact
- 63,000 threats created *per day* in 2010
- Existing malware detection techniques are insufficient
- Signatures aren't effective against new, unknown threats
- Traditional machine learning techniques like C4.5 are limited and non-adaptive

Malware Detection

- The detection approach determines how information about a program is gathered
- The detection technique determines how that information is employed

Malware Detection Approaches

- Static analysis
  - Structural and syntactical information (header data and sequences of bytes) is analyzed
  - Code is not executed, all possible branches can be analyzed
  - Disassembly is often difficult
  - Time consuming especially with obfuscated or packed code
- Dynamic analysis
  - Executes code and analyzes run-time information (contents of the run-time stack, API calls)
  - Only a single path (execution trace) is analyzed
  - High performance overhead
- Hybrid technique - combines the two previous approaches

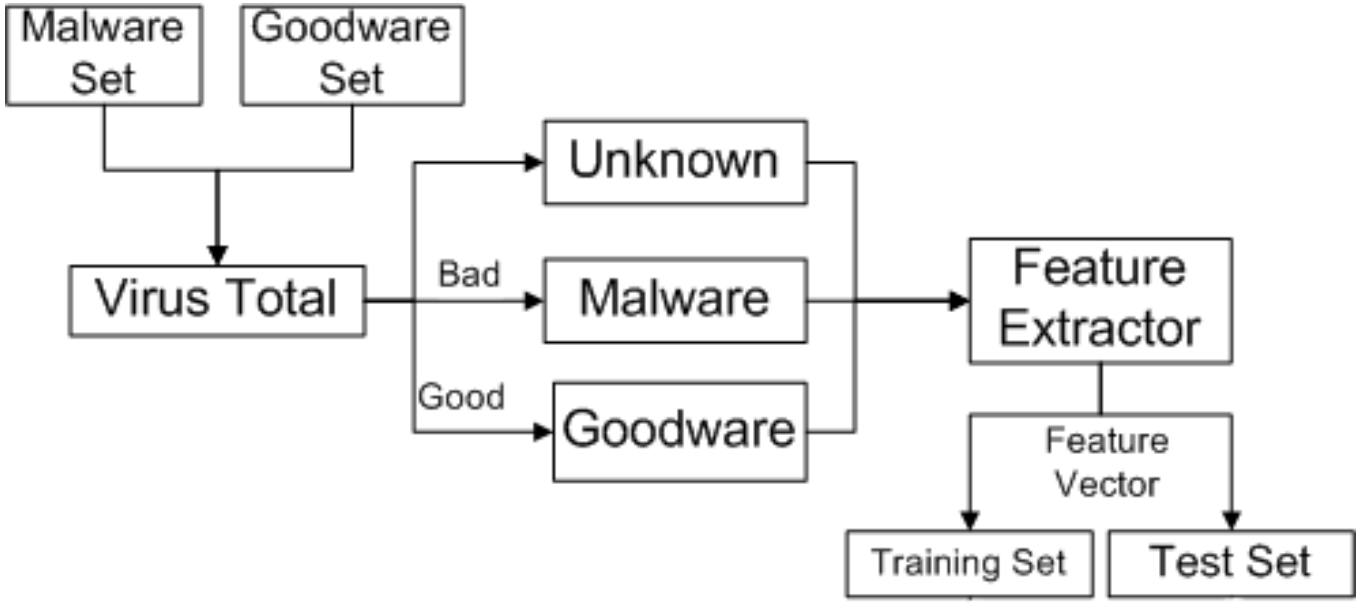
Malware Detection Techniques

- Signature-based: popular reactive technique, models malicious behavior of programs
- Anomaly-based: detects patterns in data that differ from expected behavior
- Specification-based: manually created security specifications define correct behavior

Benchmark

- 6,774 total files in the dataset
- 3,401 malicious files provided by offensivecomputing.net
- 3,373 non-malicious files from Windows XP
- 58,584 total unique imports
- 703 total unique sections

Pre-processing diagram



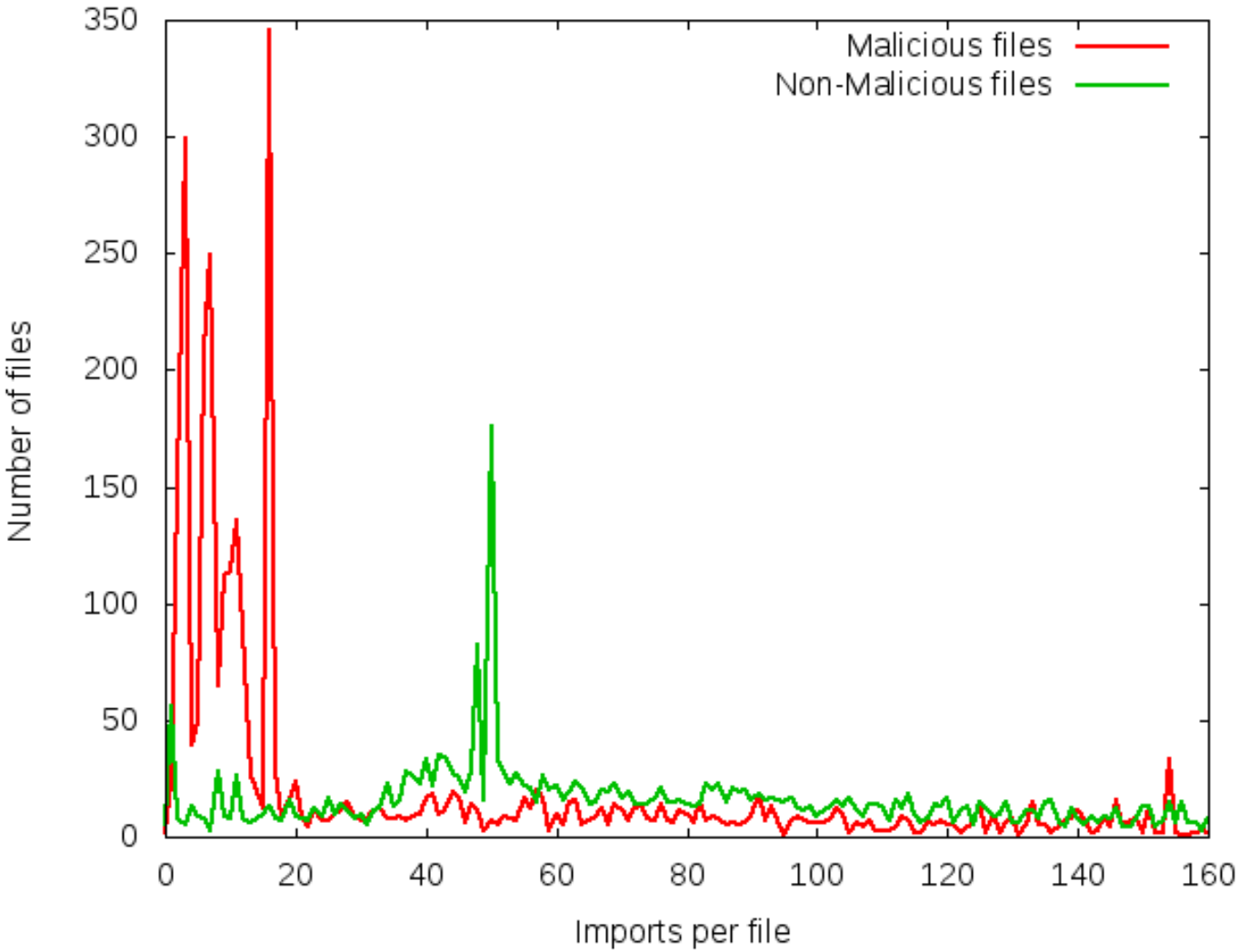
Virus Total

File name:	**70a13374**
Submission date:	**2009-10-03 10:30:14 (UTC)**
Current status:	**finished**
Result:	**34 /41 (82.9%)**

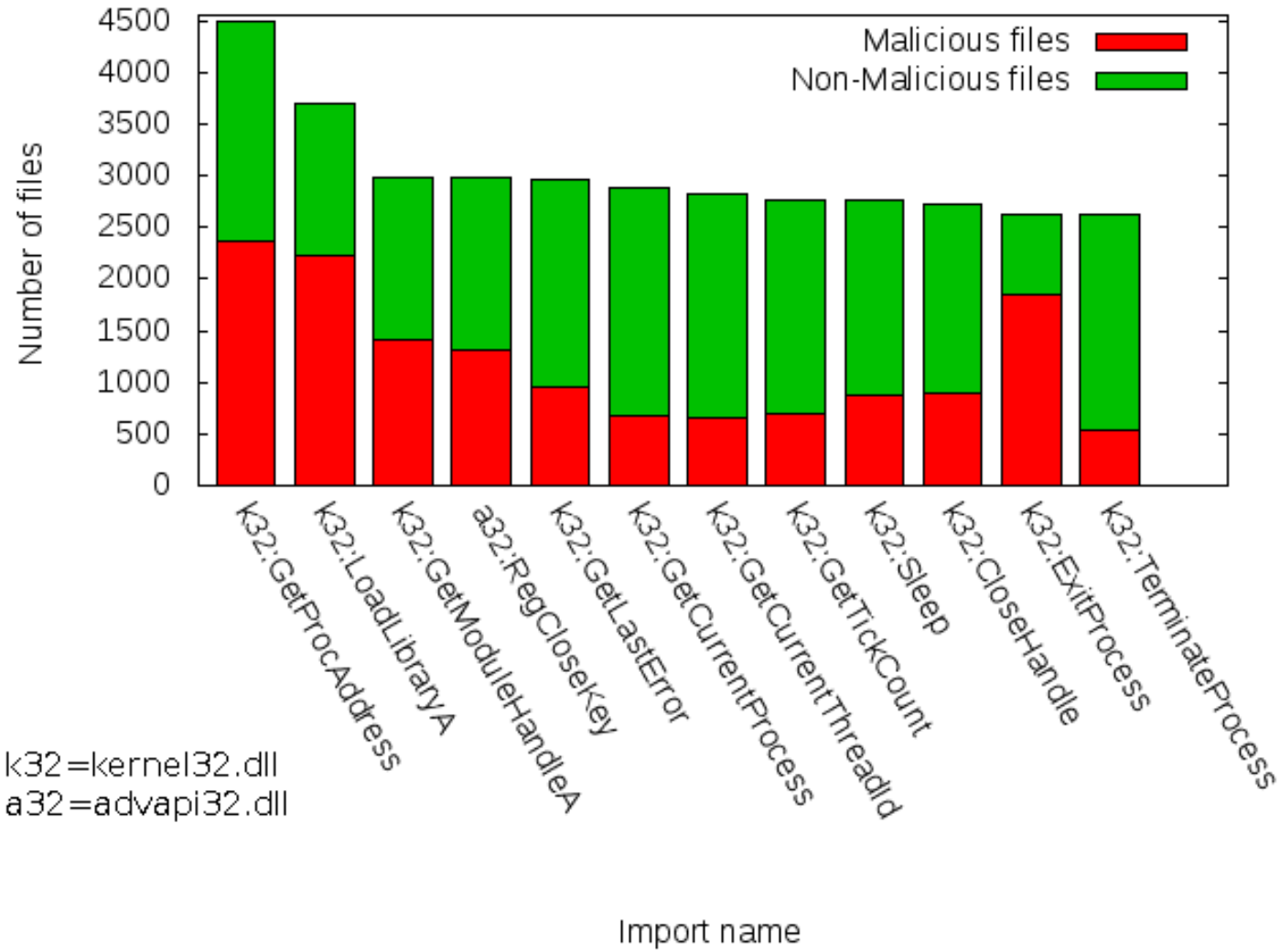
Compact

Antivirus	Version	Last Update	Result
a-squared	4.5.0.24	2009.10.03	Trojan-Spy.Banker!IK
AhnLab-V3	5.0.0.2	2009.10.02	-
AntiVir	7.9.1.27	2009.10.02	TR/Dropper.Gen
Antiy-AVL	2.0.3.7	2009.10.03	-
Authentium	5.1.2.4	2009.10.02	W32/SuspPack.M.gen!Eldorado
Avast	4.8.1351.0	2009.10.02	Win32:Spyware-gen
AVG	8.5.0.420	2009.10.03	SHeur2.AECQ
BitDefender	7.2	2009.10.03	Trojan.Spy.Bancos.NLA
CAT-QuickHeal	10.00	2009.10.03	Win32.TrojanDownloader.Banload.1
ClamAV	0.94.1	2009.10.03	-

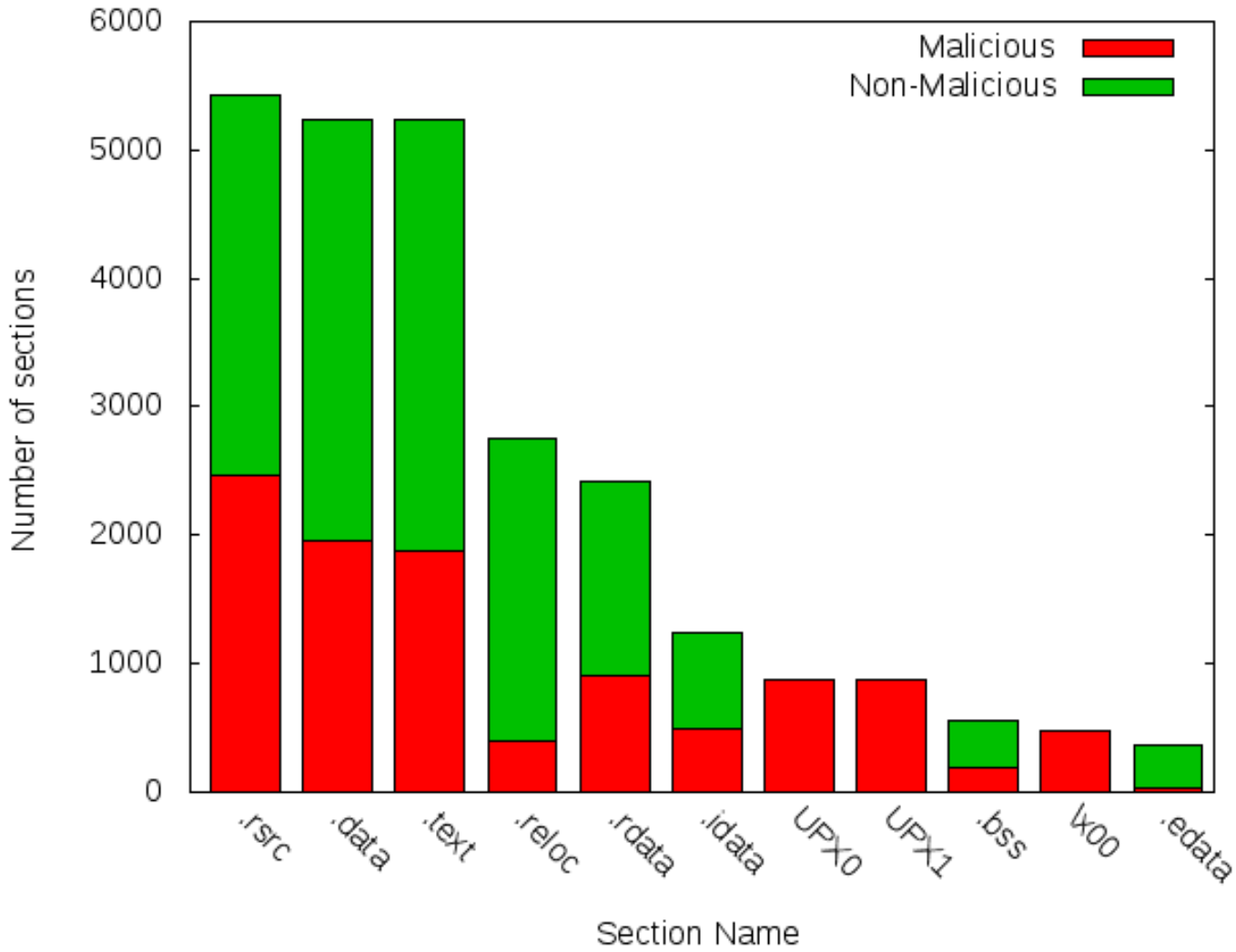
Number of imports per file



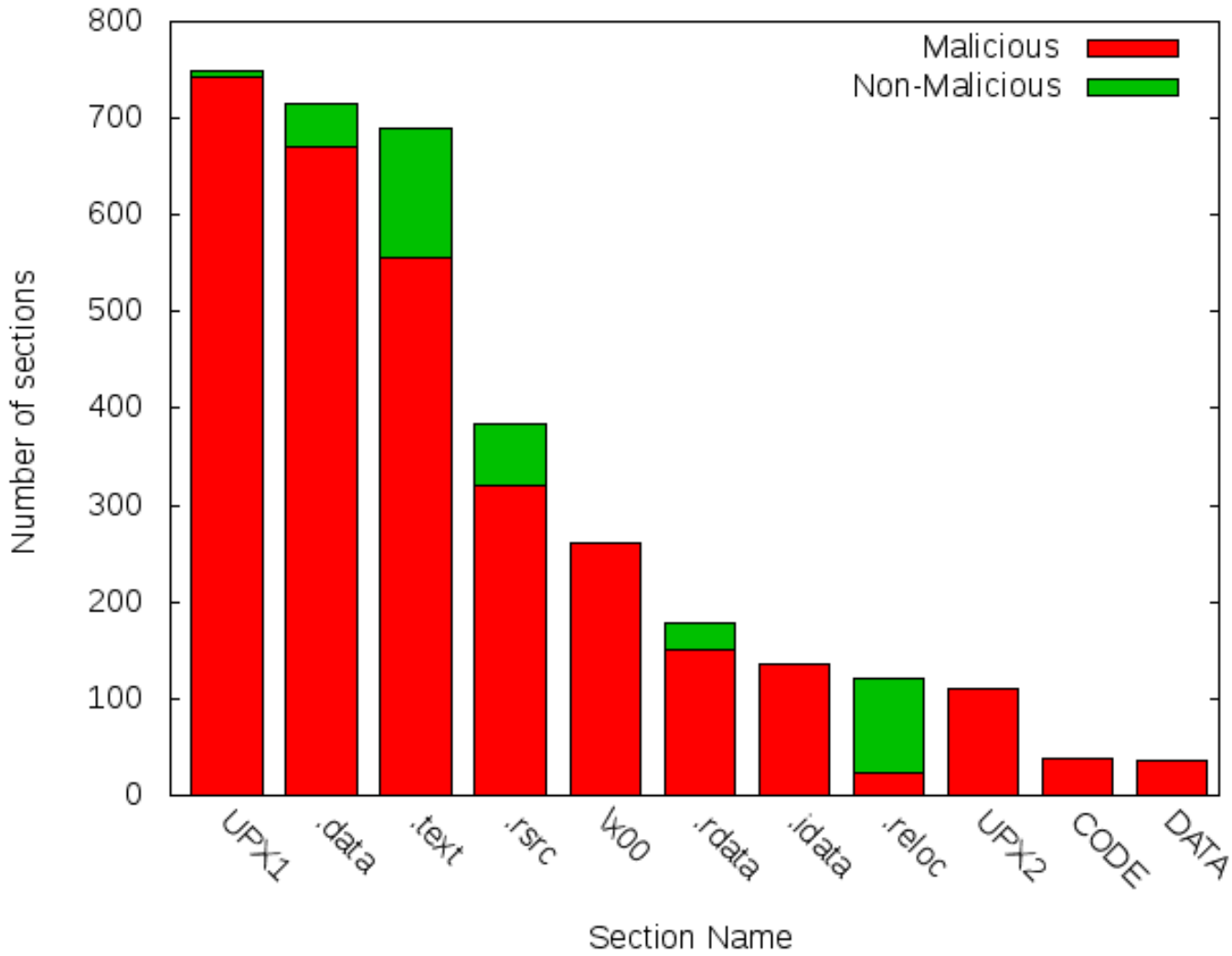
Most common imports



Most common sections



Most common high-entropy sections



Unpacked dataset

- It's trivial to distinguish packed files from unpacked files
- Unpacked dataset:
  - 854 malicious files
  - 1,048 non-malicious files
  - 18,804 unique imports
  - 126 unique sections

Dataset Noise

Two types of noise in classification problems:

- Classification noise - makes it impossible for the system to achieve 100% testing accuracy
- Attribute noise - non-predictive attributes are attributes within the problem domain that aren't useful in the system's prediction, they have no relationship to class

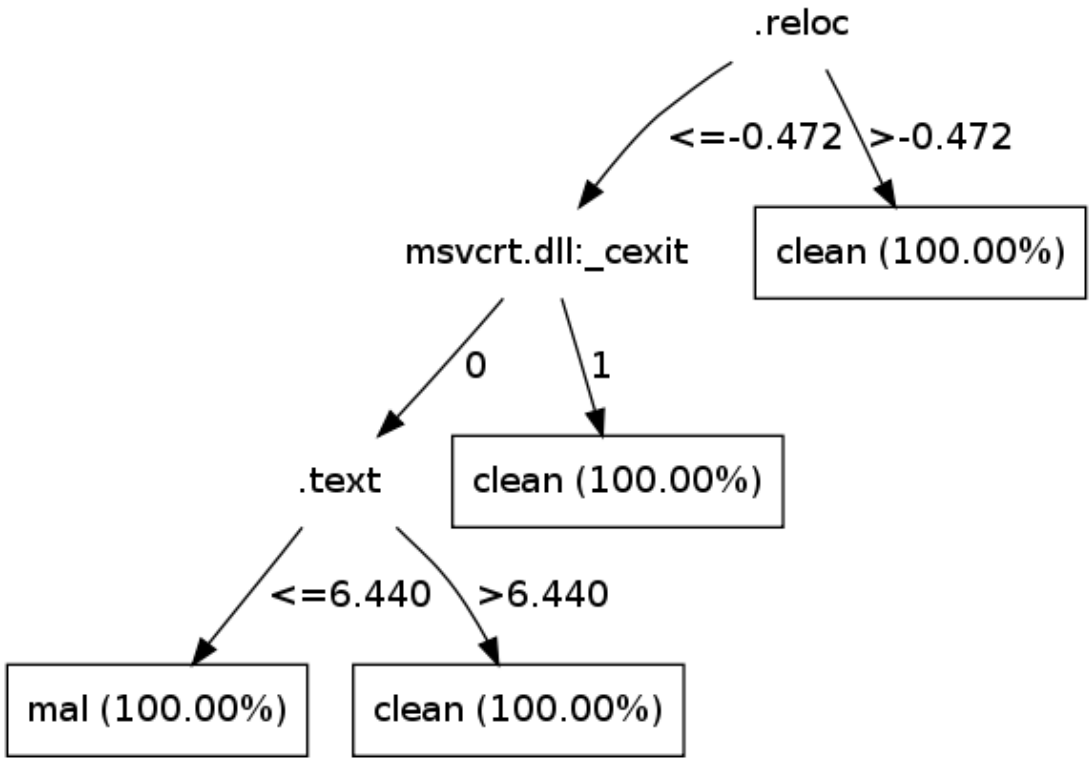
Training/Testing

- Evaluated using stratified 10-fold cross-validation
- During training:
  - The training set is repeatedly presented
  - The correct result from VirusTotal is used for reward/punishment
- The training and test sets have no files in common
- During testing, VirusTotal is employed to score the system

C4.5 algorithm

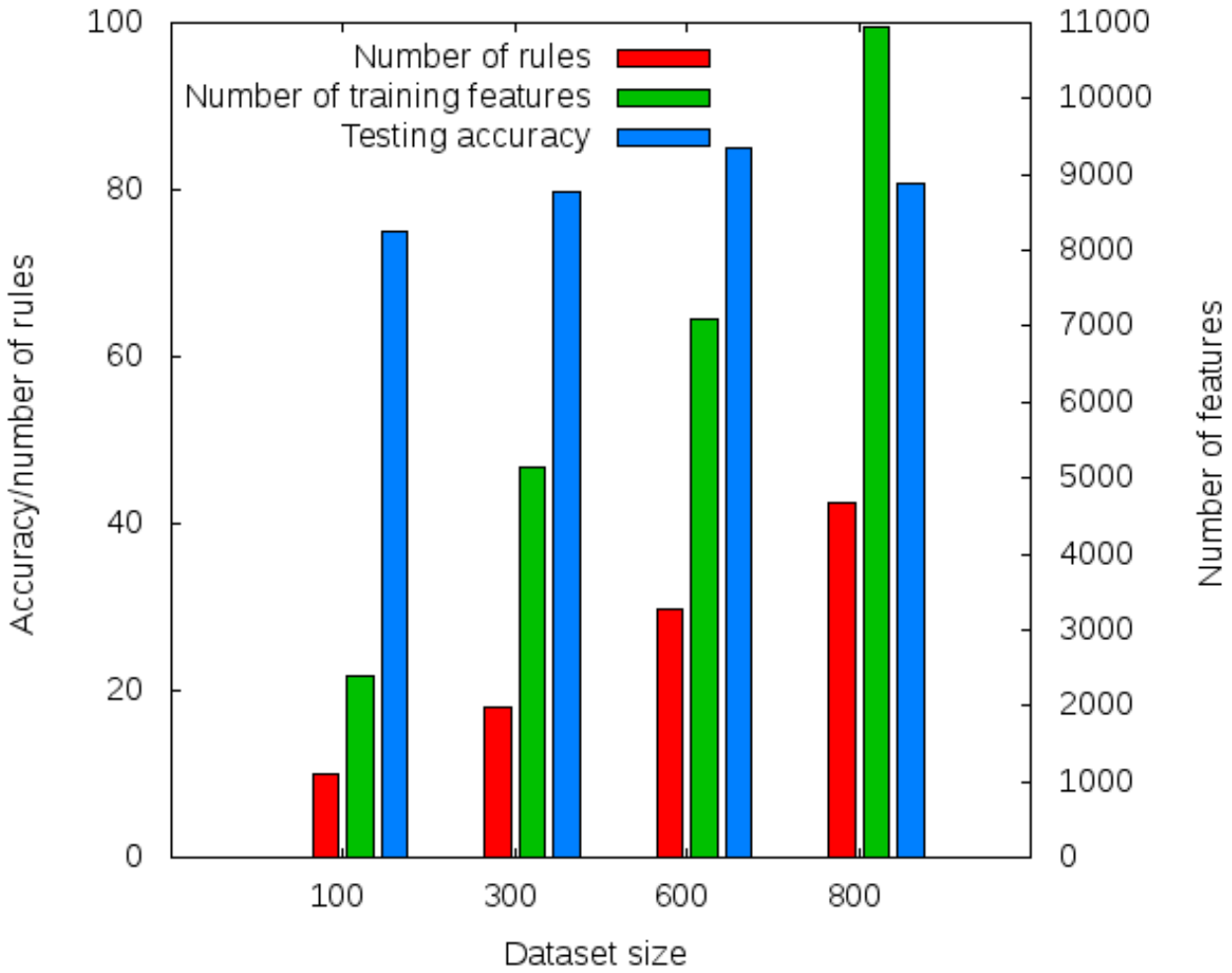
- Machine learning algorithm that builds decision trees
- Each node of the tree represents an attribute that most effectively splits the data into subsets
- Uses normalized information gain (difference in entropy) to choose an attribute

C4.5 Example tree



C4.5 results

Baseline comparison to C4.5



Evolutionary Algorithms

- An Evolutionary Algorithm is a stochastic population-based optimization meta-heuristic
- Uses mechanisms inspired by biological evolution
- Each individual in a population contains a candidate solution to the problem
- Useful in search and optimization problems

Michigan versus Pittsburgh style

Michigan Style:

- Holland's first LCS
- EA operates on the individual level
- Entire population of rules represents a solution

Pittsburgh Style:

- A population consists of variable length rule sets
- EA operates on the level of an entire rule set
- Each rule set is a potential solution

Online versus offline learning

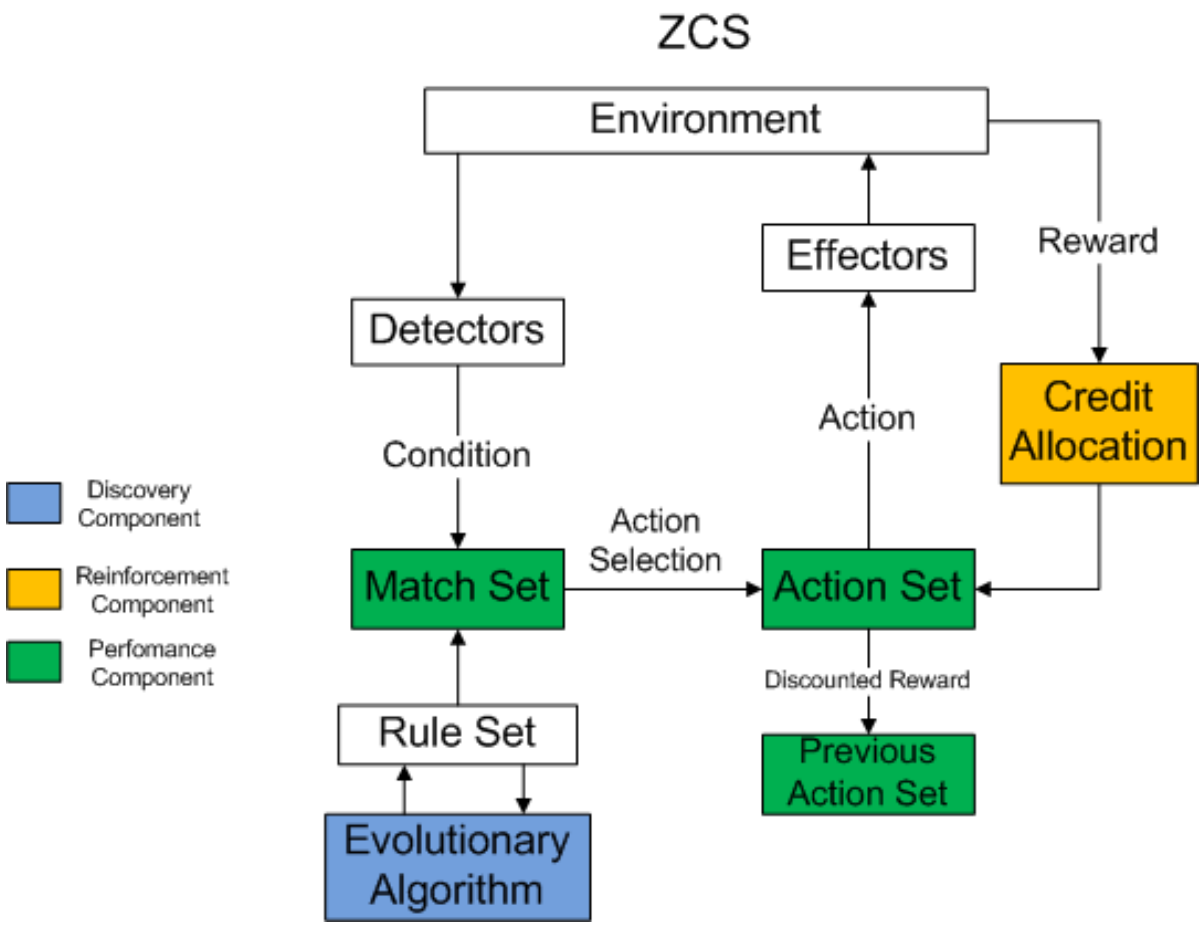
Offline Learning (batch learning):

- All training problems are presented simultaneously
- Results in a rule set that does not change over time
- Pittsburgh style LCSs are usually applied in this method
- Used in data mining problems

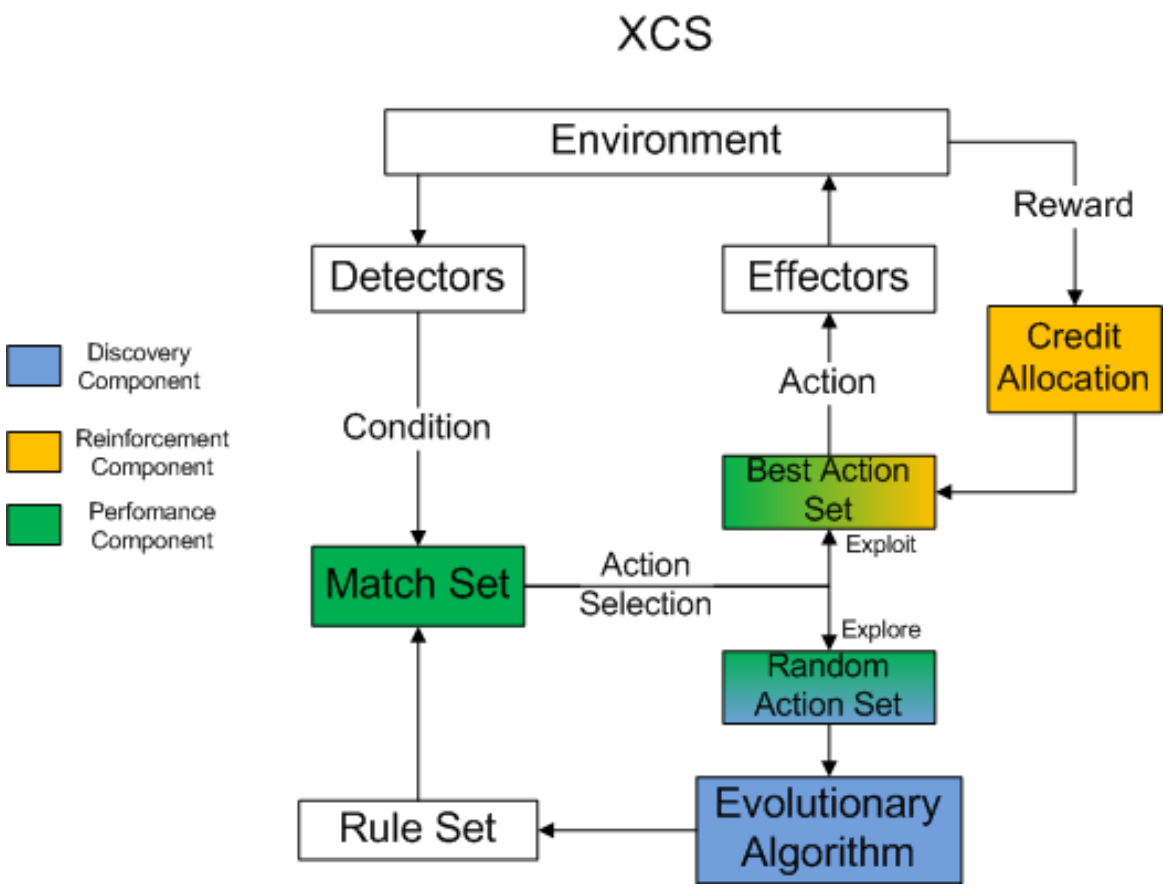
Online (incremental) learning:

- Problem instances presented individually
- Rule set evolves over time with each new observation
- Typical of Michigan style LCS (though Pittsburgh can use online learning)

ZCS


ZCS
Environment
Effectors
Reward
Detectors
Credit Allocation
Action
Condition
Discovery Component
Reinforcement Component
Perfomance Component
Action Selection
Match Set
Action Set
Discounted Reward
Rule Set
Previous Action Set
Evolutionary Algorithm

XCS

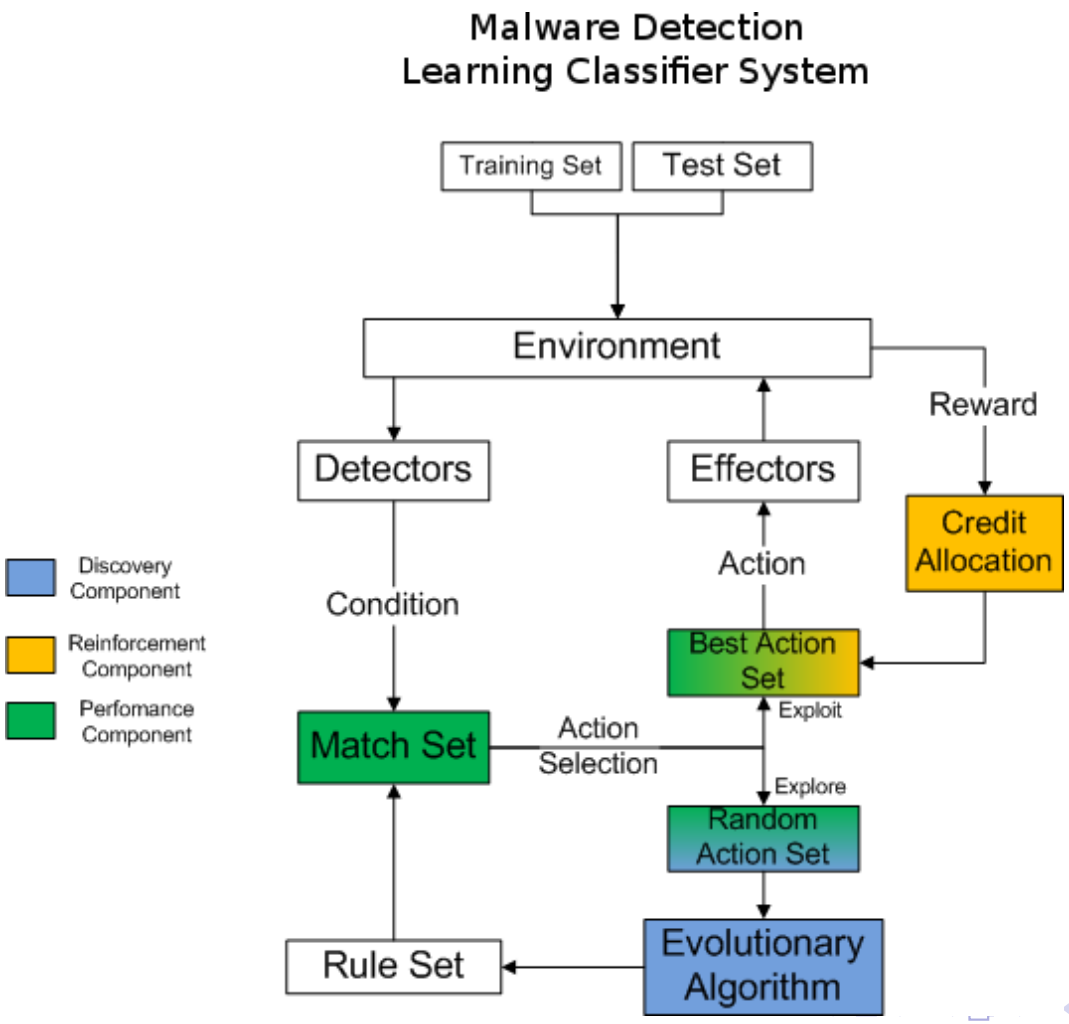

XCS
Environment
Reward
Detectors
Effectors
Credit Allocation
Action
Condition
Best Action Set
Exploit
Match Set
Action Selection
Explore
Random Action Set
Evolutionary Algorithm
Rule Set
Discovery Component
Reinforcement Component
Perfomance Component

Overgeneral Classifier Problem

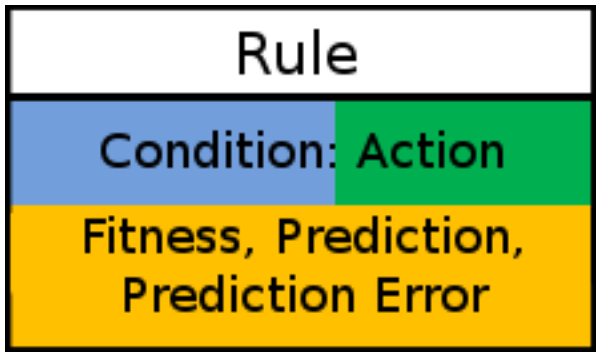
- *General classifier* - condition matches many problem instances
- *Greedy classifier* - fitness depends on the magnitude of the reward it receives
- *Overgeneral classifier* - advocates desired action only in certain problem cases
- The problem occurs when rules act correctly in some states but incorrectly in others
- More specialized rules may be outvoted by overgenerals
- Strong overgeneral rules are unreliable, but outweigh reliable rules during action selection

Malware Detection LCS diagram

The system is based on XCS



Rule (classifier) definition



- Condition - truth function which is satisfied by inputs
- Action - rule advocates an action for the system to perform
- Fitness values - accuracy of prediction of environmental reward

LCS Training/Testing

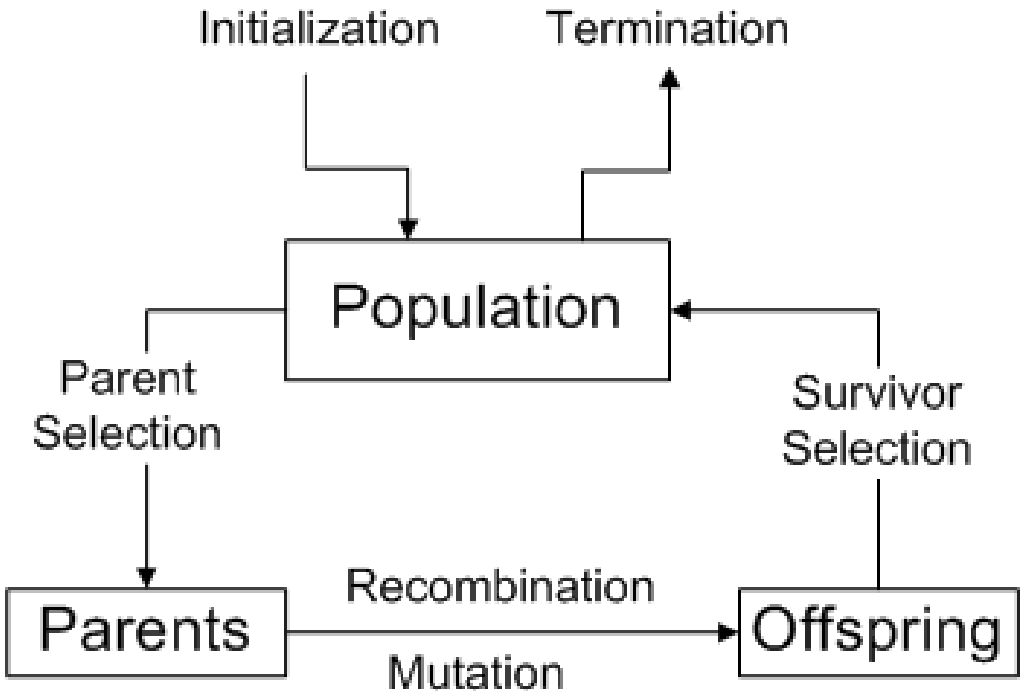
- Training set used to evolve rules to identify the malicious samples
- Current population is run against the test set periodically, to evaluate generalization of the ruleset
- Correct results are known during testing to score accuracy, but no learning takes place

Initialization Methods

3 methods:

- Random - Entire population created by generating random trees (conditions) and actions
- Covering - Rules created by the covering operator, population starts empty
- C4.5 - Each leaf node in a decision tree converted into a single rule, specialized by adding nodes

Rule Evolution cycle



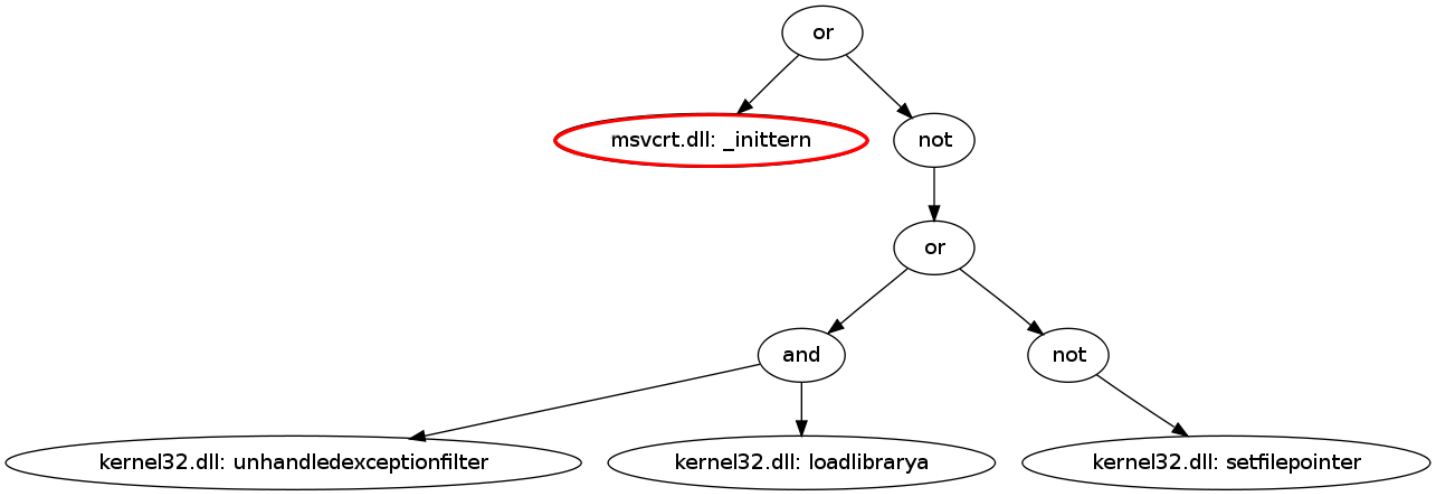
Evolutionary Pressure in XCS

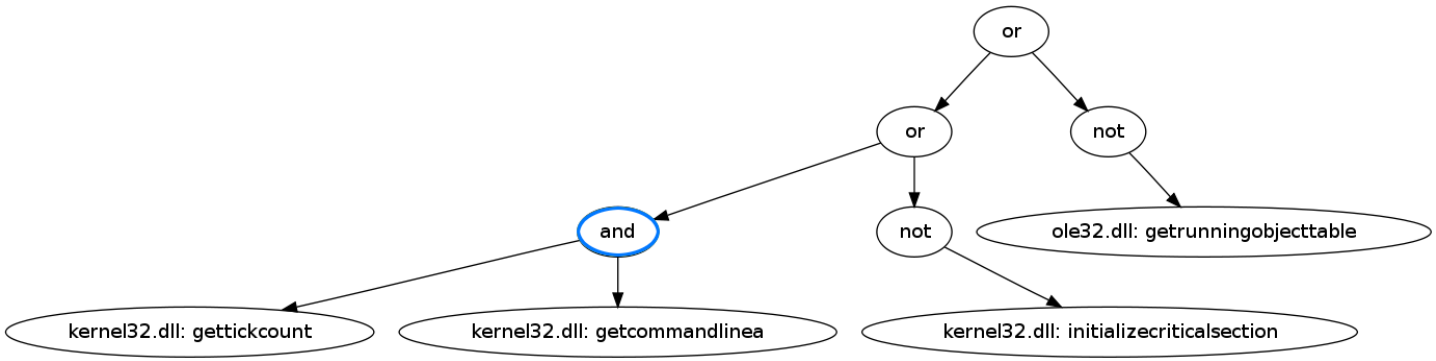
- XCS's generalization pressure is overcome by a strong fitness pressure (the pressure towards higher accuracy)
- *Proportionate selection* - the smaller the fitness differences in the population, the smaller the fitness pressure
- In XCS, rule fitness is derived from a scaled, action set-relative accuracy
- *Tournament selection* - based on fitness rank, not relative fitness differences
- Tournament selection does not suffer from fitness scaling or from small differences in accuracies

Tournament Size in XCS

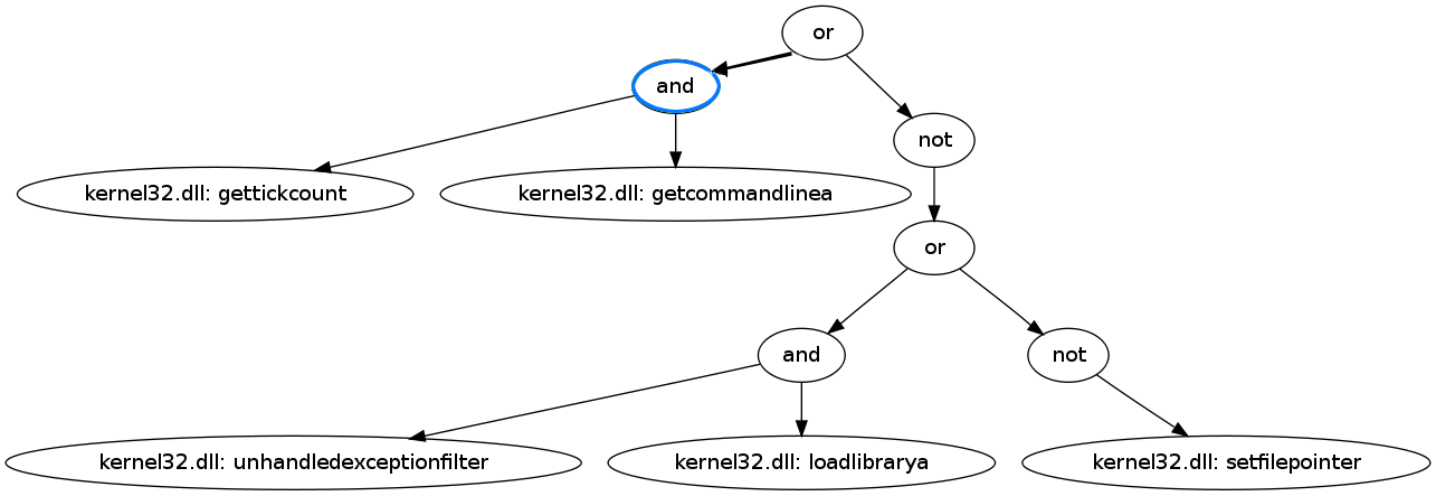
- In XCS the EA acts on the action set
- In XCS, action set sizes can vary significantly
- A change in the tournament size changes the strength of the selection pressure applied
- A relatively strong selection pressure, which adapts to the current action set size, is required
- XCS tournament size is dependent on the current action set size, ensuring equal selection pressure

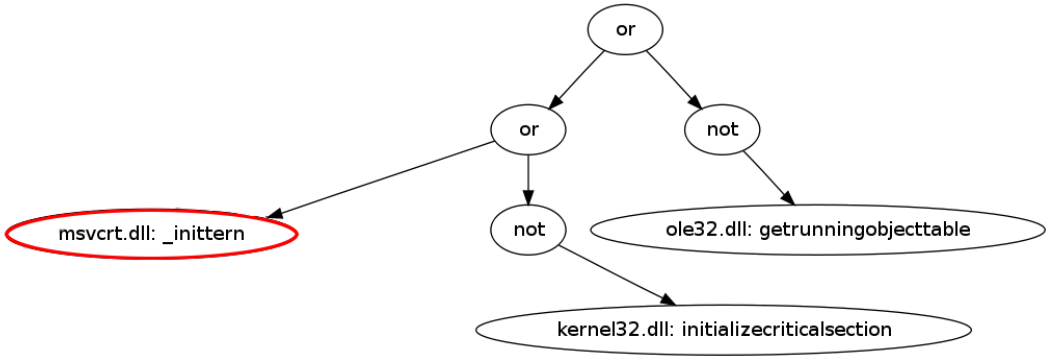
Crossover example - parents





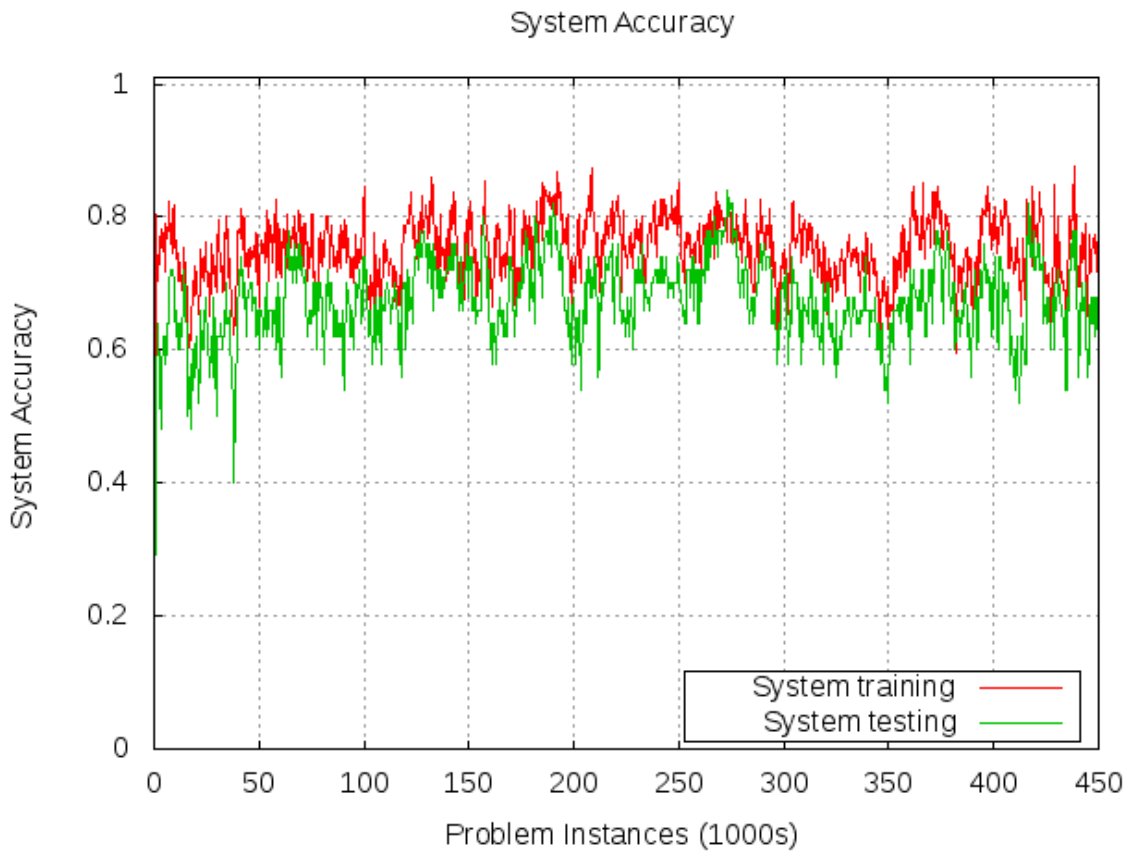
Crossover example - offspring



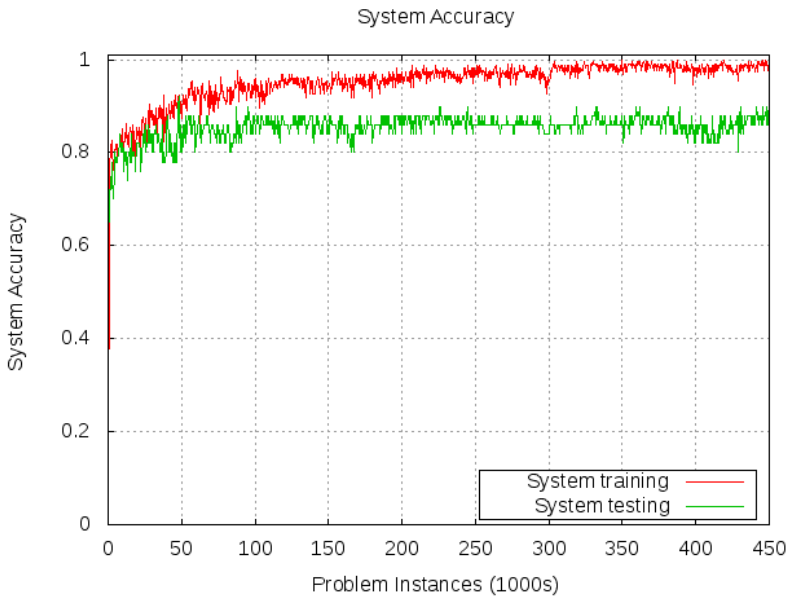


Covering-random deletion cycle

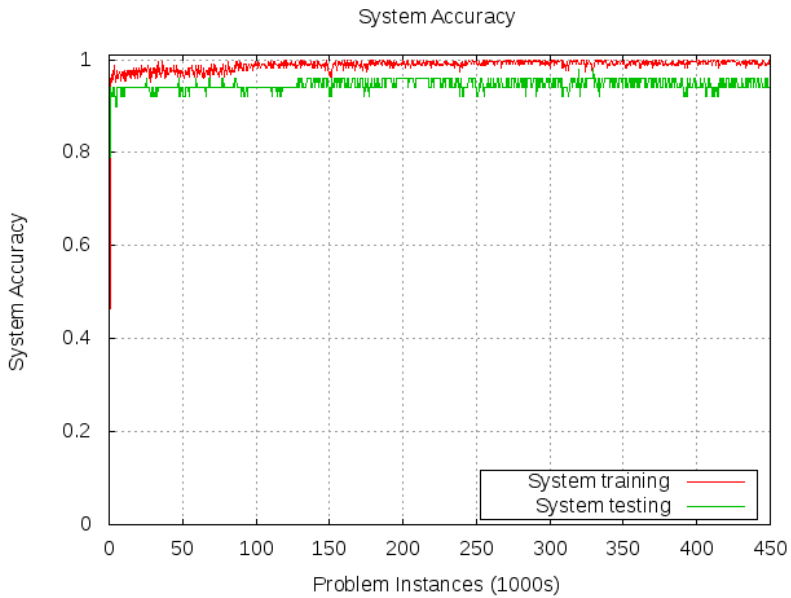
- Rules' conditions are too specific
- Too small a population size (10)



Adequate population size (30)



(a) random initialization

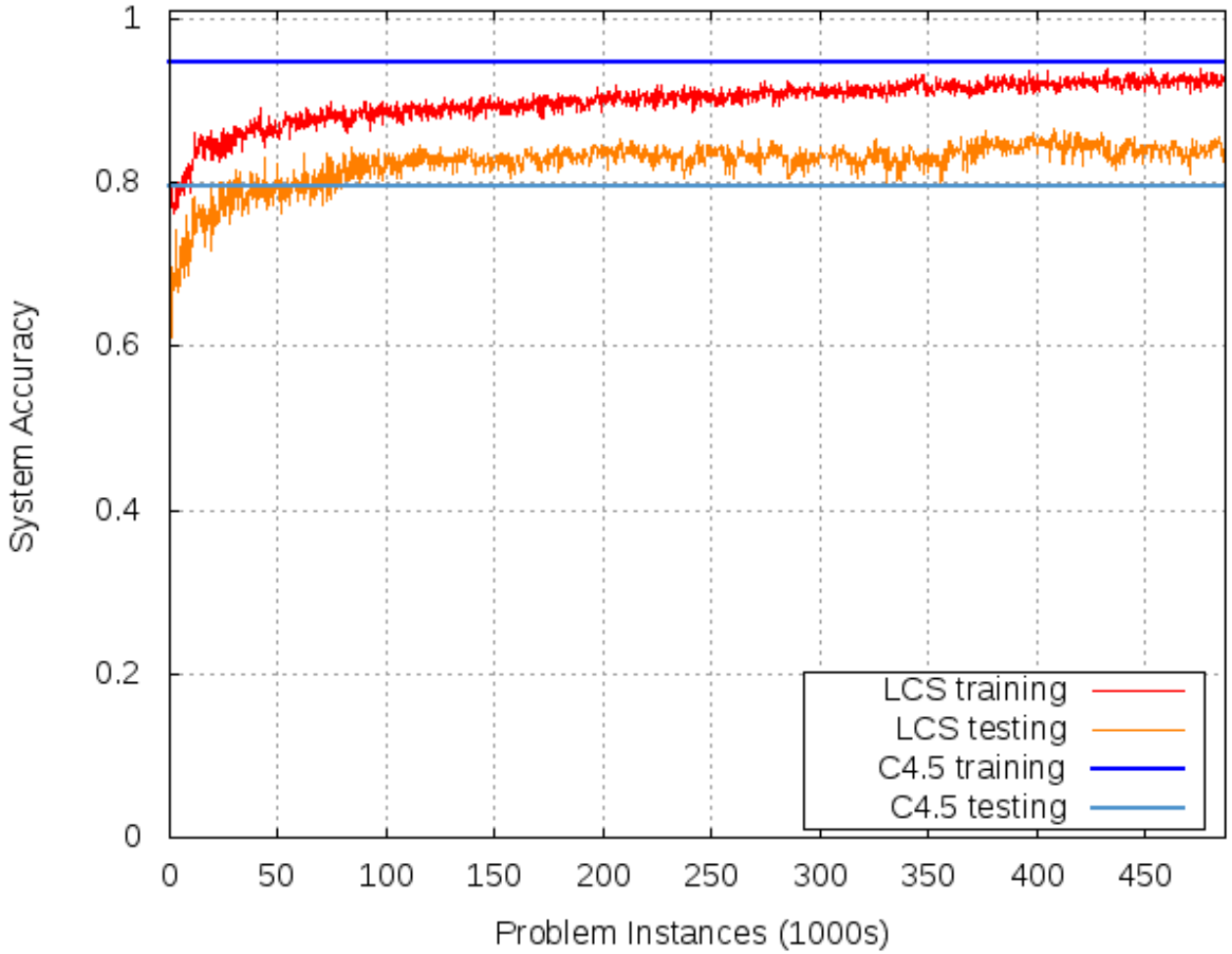


(b) C4.5 initialization

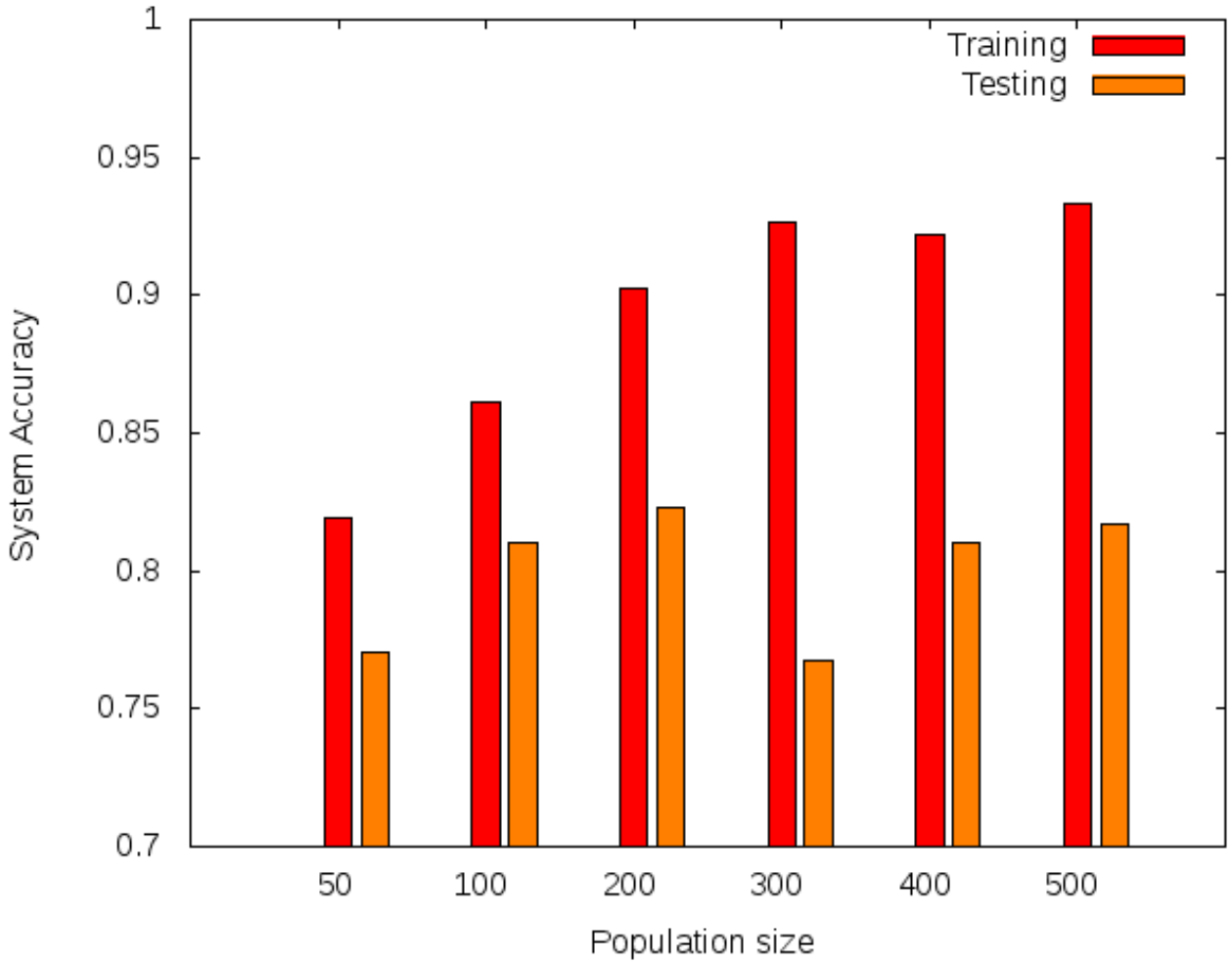
Malware family study

- Too small a population size (10), covering-random deletion loop occurred
- The C4.5 decision tree performed better than the LCS at a lower population size
- Increasing the population to 20 and 30 allowed the LCS to perform as well as C4.5 (and in one case out-train it)
- C4.5 initialization method converged quicker than other methods

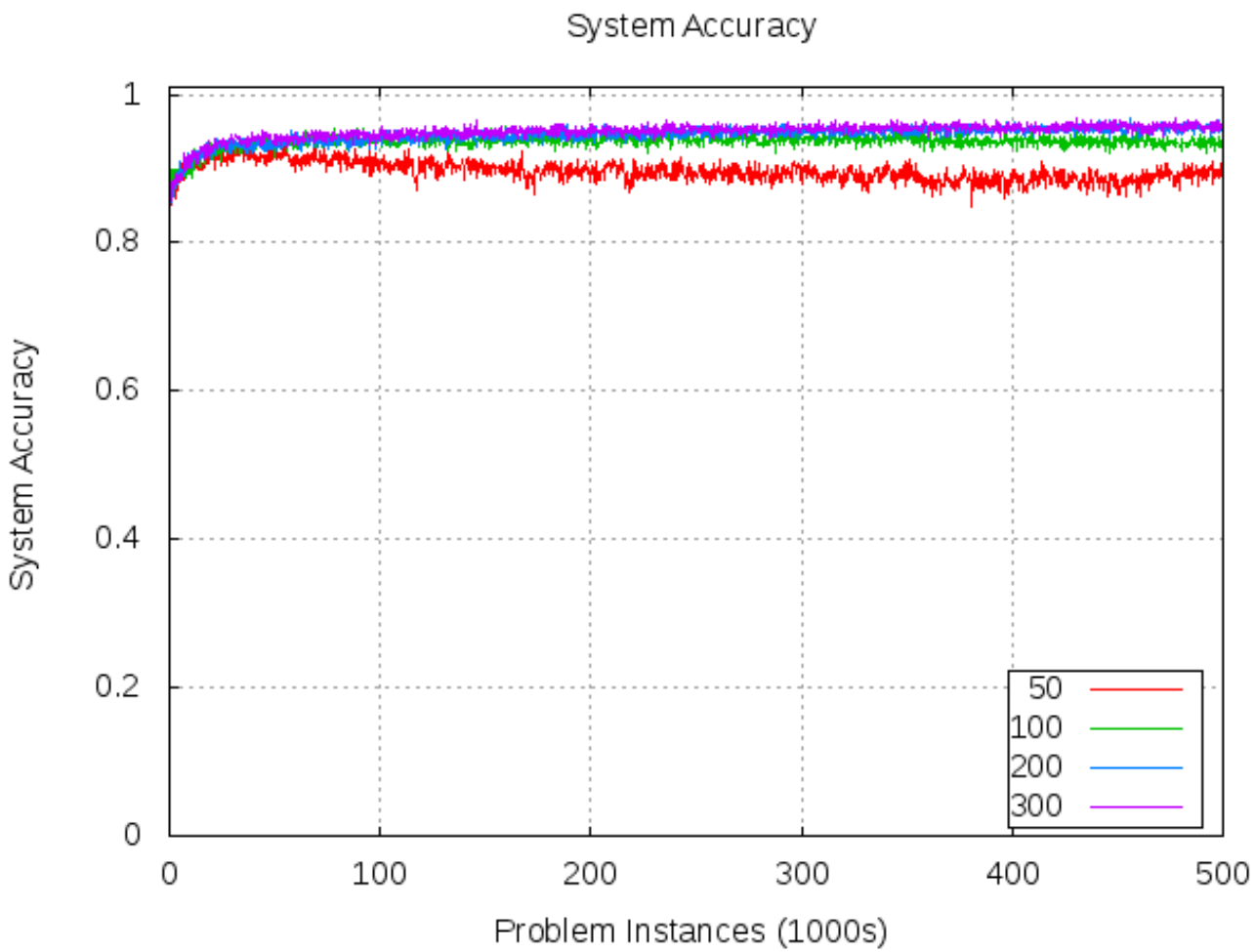
Evolution trends



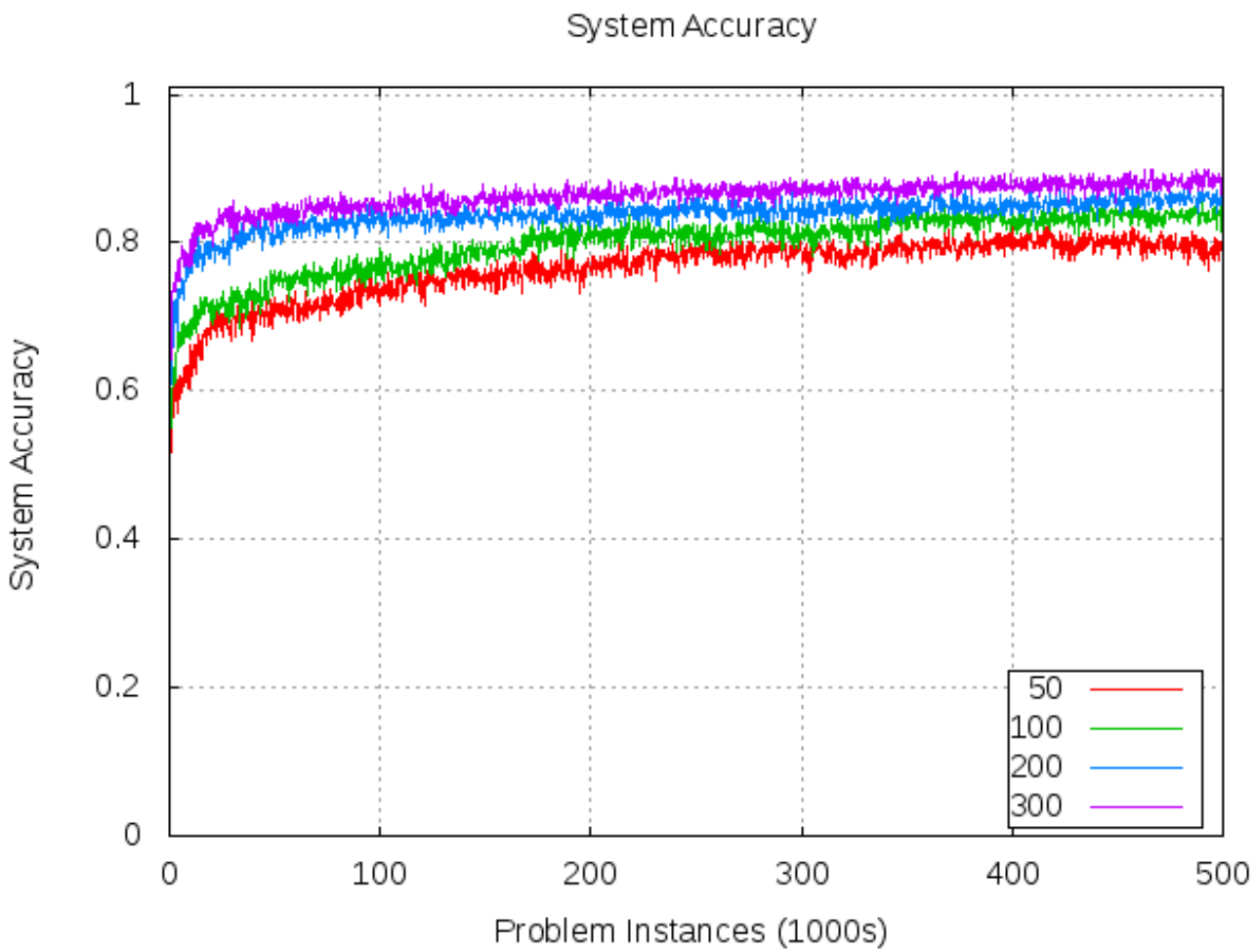
LCS population size tuning



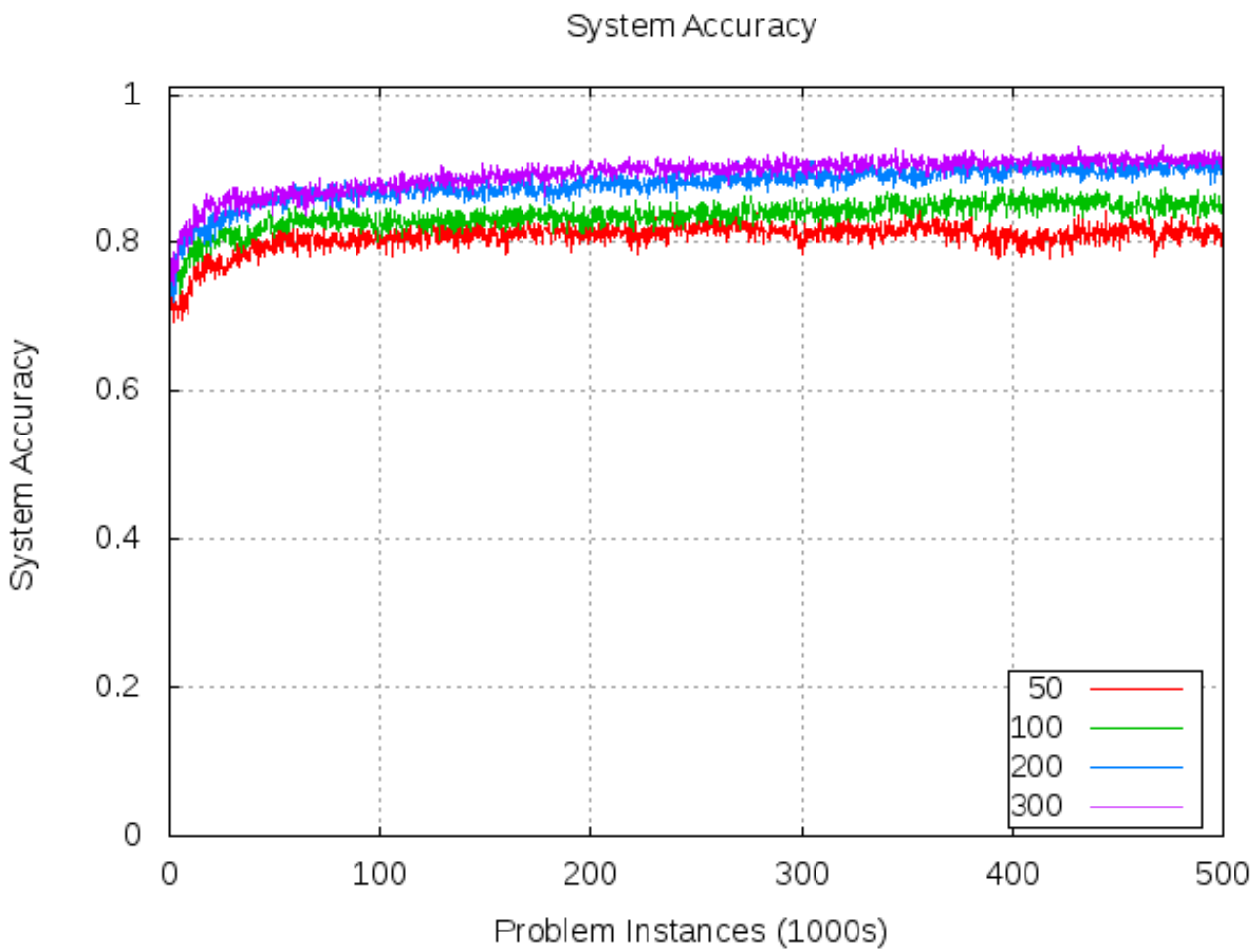
Population size study - C4.5 initialization



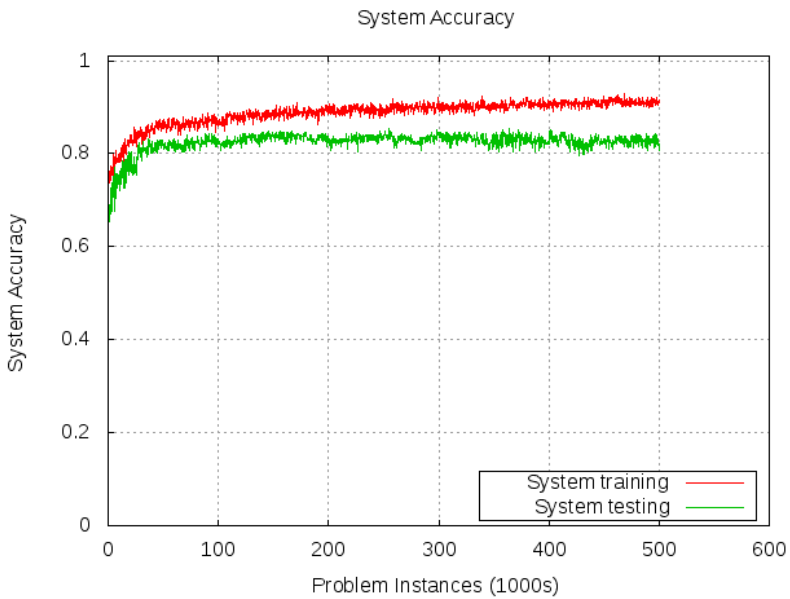
Population size study - Random initialization



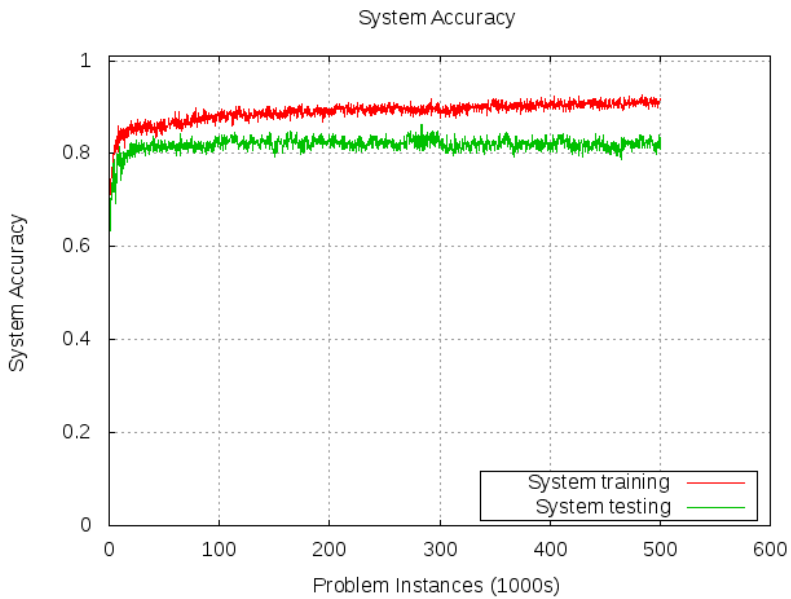
Population size study - Covering initialization



Offspring size study

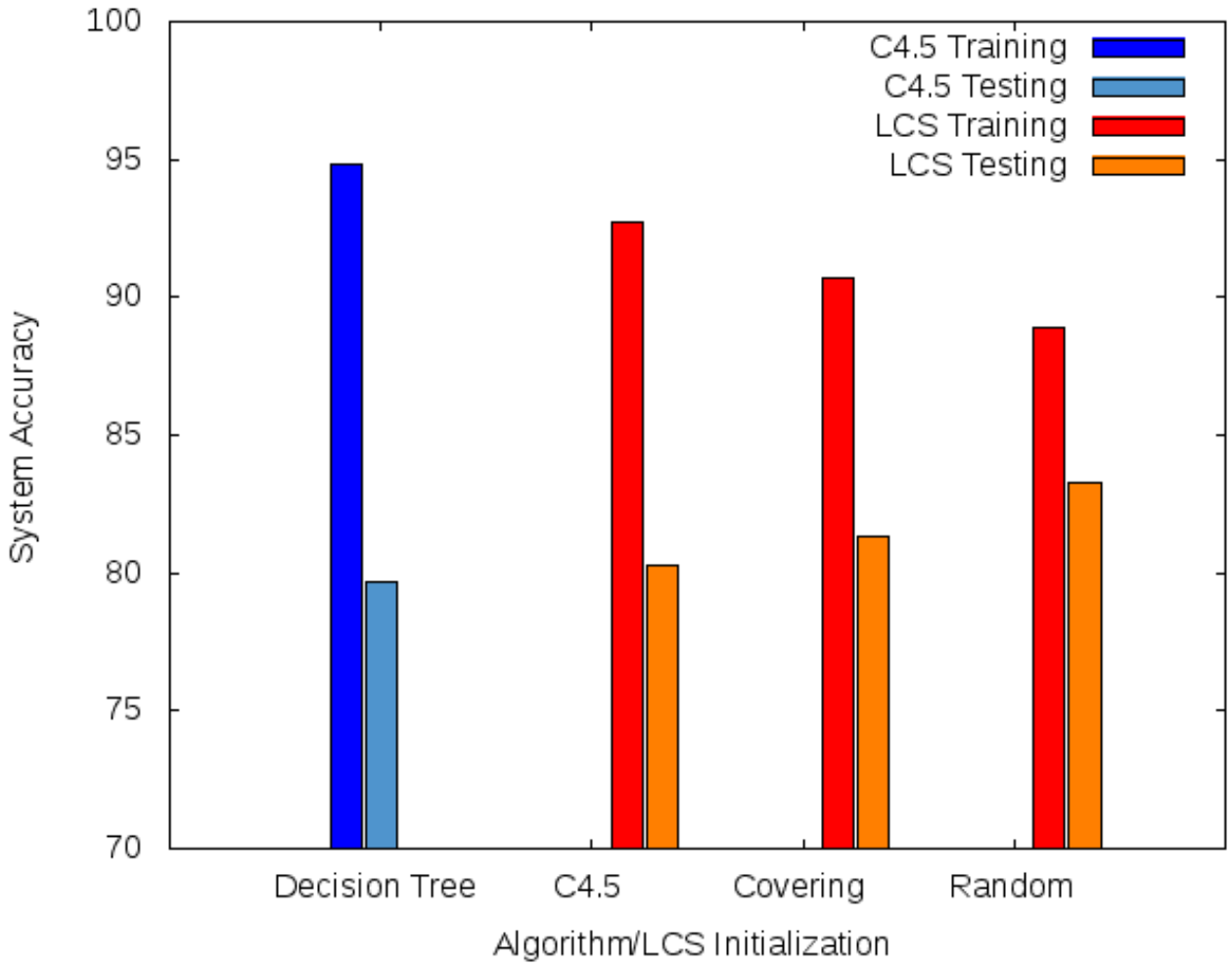


(c) Offspring size = 1



(d) Offspring size = 2

C4.5 vs. LCS


100
95
90
85
80
75
70
System Accuracy
C4.5 Training
C4.5 Testing
LCS Training
LCS Testing
Decision Tree
C4.5
Covering
Random
Algorithm/LCS Initialization

Conclusions - Initialization methods

- C4.5 initialization has promising results, in some cases outperforming the decision tree it was seeded from
- Covering initialization can achieve similar performance as C4.5 but takes longer to converge
- Random initialization is slower to converge and has higher performance variance than C4.5
- Generalization of the LCS was better than that of C4.5 decision tree, but there is still some overfitting between the training and testing results
- Overfitting indicates that the system's learning is specific to the structure of the training set and generalization is not optimal

General Conclusions

- C4.5 typically trains to a very high accuracy, but lacks in generalization of testing data
- LCS doesn't suffer from C4.5's feature dimension limitation
- LCS has the potential to outperform more traditional non-adaptive techniques such as C4.5
- LCS is sensitive to certain parameter values, such as population size
- Tuning the LCS is required for optimal performance, as parameters interact with each other
- Noise in the dataset and problem space accounted for a large variance in performance
- Larger datasets were more diverse and thus more difficult to classify

Future work

- Expand feature set of PE files
- Investigate using dynamic as well as static analysis
- Compare with larger and more diverse datasets
- Compare LCS with other machine learning techniques such as random forest and C5.0
- LCS parameter tuning needed to obtain optimal performance
- Using a Fuzzy LCS may allow for better generalization
- Investigate why overfitting doesn't appear in individual experiments