RC24442 (W0711-220) November 28, 2007
Computer Science

IBM Research Report

Policy Evolution with Genetic Programming

Yow Tzu Lim[1], Pau Chen Cheng[2], Pankaj Rohatgi[2], John Andrew Clark[1]

[1]Department of Computer Science
University of York
UK

[2]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
USA

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Policy Evolution with Genetic Programming

Yow Tzu Lim*, Pau Chen Cheng†, Pankaj Rohatgi† and John Andrew Clark*
*Department of Computer Science, University of York, UK.
Email: {yowtzu, jac@cs.york.ac.uk}
†Department of Security and Privacy, IBM Hawthorne Watson Research Labs, USA.
Email: {pau, rohatgi@us.ibm.com}

Abstract

A good deal of work has been carried out in the area of security policies and security policy modelling. In the early days a policy was a set of simple rules with a clear intuitive motivation that could be formalised to good effect. Bell-LaPadula [1] MLS policy is a good example. However the world is now much more complex. Subtle risk decisions may often need to be made and people are not always adept at expressing rationale for what they do. In this paper we investigate how statements of policies can be derived automatically from examples of decisions made. This allows us to automatically discover a policy that may not formally have been documented, or else extract an underlying set of requirements by interpreting user decisions to posed "what if" scenarios. The chosen approach of policy inference is Genetic Programming (GP). Two proof of concept experiments on Bell-LaPadula [1] MLS policies and Fuzzy MLS policies [2], [3] have been carried out. Each decision action in a policy is considered as a classic set and the GP is used to search for the necessary conditions to be a member of that set. The results show that this approach is promising and very reasonable policies can be inferred if the examples in training set are sufficiently diverse. Furthermore, the learning process is largely independent of the parameter settings. Finally, to relax the requirement for diverse examples in a training set, we consider each decision action as a fuzzy set and try to learn its fuzzy membership function. Again, our result shows that such fuzzy model can achieve comparable accuracy with their corresponding classic models yet depends less on the diversity of training examples.

Index Terms

Genetic Programming, Policy Inference, Security Policy, MLS.

I. Introduction

In computer systems, a security policy is essentially a set of rules specifying the way to secure a system for the present and the *future*. Forming a security policy is a challenging task: the system may be inherently complex with many potentially conflicting factors. Traditionally security policies have had a strong tendency to encode a static view of risk and how it should be managed (most typically in a pessimistic or conservative way) [12]. Such an approach will not suffice for many dynamic systems which operate in highly uncertain, inherently risky environments. In many military operations, for example, we cannot expect to predict all possible situations.

Much security work is couched in terms of risk but in the real world there are benefits to be had. In military operations you may be prepared to risk a compromise of confidentiality if not doing so could cost lives. There is a need for operational flexibility in decision making, yet we cannot allow recklessness. Decisions need to be defensible and so must be made on some principled basis. People are typically better at making specific decisions than in providing abstract justification for their decisions. It is very useful to be able to codify in what a "principled basis" consists since this serves to document "good practice" and facilitates its propagation.

The above discussion has been couched in terms of human decision making. In some environments the required speed of system response may force an automated decision. Such automated decisions must also be made on a "principled basis", and some of these decisions may be very tricky. Automated support must be provided with decision strategies or rules to apply.

Research was sponsored by US Army Research laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF–06–3–0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

In this paper we investigate how security policy rules can be extracted automatically from examples of decisions made in specified circumstances. This is an exercise in *policy inference*. The automation aspect of the inference is doubly useful: automated inference techniques can discover rules that humans would miss; and policies can be dynamically inferred as new examples of tricky decisions become available. Thus the current policy can evolve to reflect the experience of the system.

For example, if a human determines what the proper response should be based upon the information available, either in real-time or post facto, a conclusion is drawn that similar responses should be given under similar circumstances. Essentially, we attempt to partition the decision space such that each partition is associated with a response that is commensurate with the risk vs. benefit trade-off.

In practice, different decision makers may come to different decisions in the same circumstances, particularly if the decisions are tricky. Decision makers may use data that is not available to the inference engine to reach a decision, or else one decision-maker may simply have a different appetite for risk. Any inference technique must be able to handle sets of decision examples that do not seem to be entirely consistent. The chosen inference approach is Genetic Programming (GP). It is a specific form of guided search that has seen success in a variety of domains.

This paper documents the initial results of two proof of concept experiments: inferring Bell-LaPadula MLS and Fuzzy MLS policies from examples using GP. Each decision action in a policy is considered as a classic set and GP is used to search for the necessary conditions to be a member of that set. The results show that this approach is promising.

The organisation of this paper is as follows: Section II presents a brief introduction to evolutionary algorithms, in particular Genetic Programming (GP). Section III documents the related work in this domain and Section IV describes the general design of the experimental framework. Section V and VI present the experimental designs and results obtained in inferencing MLS Bell-LaPadula and Fuzzy MLS policies. Section VII presents the experimental designs and results obtained in using a modified fuzzy set concept to address the issues discovered in the previous experiments. Finally we conclude the report with a summary of the experiment results and possible future work.

II. Evolutionary Algorithms and Genetic Programming

Evolutionary algorithms (EAs) are problem solving techniques inspired from natural selection[1] and the Lamarckian hypothesis[2]. An initial population of individuals, which represent candidate solutions to the problem in question, are generated. There are a variety of ways of doing this but some form of randomised approach is normally used. Each individual can be associated with a "fitness" that measures how well each candidate solves the problem. In many applications this is formed by the application of some defined function, $f(sol)$ to the candidate solution, sol. In others, the "real world" acts as the cost function. For example, it is far easier to see how much power a program consumes by running the program and measuring its power consumption than it is to derive and use a predictive model of power consumption.

The fitness of the initial population are evaluated and the population is then subject repeatedly to the three evolutionary operators:

- Selection: individuals are selected for breeding according to fitness (natural selection);
- Crossover: selected individuals are bred; parts of each solution are exchanged to form two new individuals;
- Mutation: elements within a solution are perturbed in some way. This serves to diversify solution elements in the population. Given an initial population, repeated application of selection and crossover alone might otherwise not be able to reach parts of the search space.

This evolutionary process produces populations of individuals (candidate solutions) that are increasingly better suited to the environment (problem). Commonly, the stopping criterion for EAs is either that a "good enough" solution has been found or that a preset number of generations have been produced. The evolutionary process is depicted in Figure 1.

[1]Natural selection states that individual that are best adapted to the environment have better chance to survive and reproduce for next generation.

[2]The Lamarckian hypothesis states that learnt characteristics can be inherited between generations. This is a controversial subject in the biological world (and of very limited application) but we are not bound by the biological reality in the computational world.

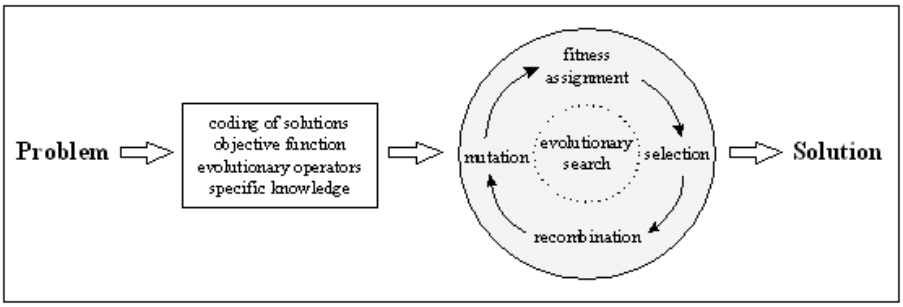
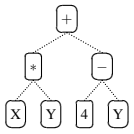


Fig. 1. Solving problem using Evolutionary Algorithms (EAs) [15]

There are a variety of EAs. Representations of solutions vary a lot. In most applications solutions are represented as linear lists of elements of the solution. GP has traditionally used tree representations (originally trees of LISP S-expressions) [4]. An example is shown in Figure 2.



$$(+\ (*\ X\ Y))\ (-\ 4\ Y))$$

Fig. 2. An individual example in GP and its corresponding LISP S-Expression

The nodes in the LISP tree can be classified into two groups: the terminal set T and function set F. The terminal set T consists of constants and variables (the leaf nodes) and the function set F consists of functions, operators and statements (the non-leaf nodes). An example terminal set T and function set F sufficient to allow description of the LISP tree in Figure 2 is given below as:

$$\begin{aligned} T &= \{X, Y\} \bigcup \{1, 2 \ldots 100\} \\ F &= \{+, -, \times, \div, \mod\} \end{aligned}$$

In choosing these sets, there are two properties that must be ensured: the *sufficiency* property and *closure* property [13]. The sufficiency property requires that the target solution for the problem in question can be represented with the elements in the sets. The closure property refers to the elements in the function set F being able to take any value it may receive as input, including all the elements in the terminal set T and any return values from other functions or operators. Practically, it is important to keep both sets small to prevent the search space from becoming too large [6].

There are two standard evolutionary operators in GP: *crossover* and *mutation*, which are defined as follows [4]:

1) Crossover is performed on two trees. A sub-tree in each tree is chosen randomly and swapped with the other. An example of crossover is depicted in Figure 3. The marked sub-trees in the two trees are swapped with each other, resulting in two new trees.
2) Mutation is performed on one tree. A node in the tree is chosen randomly and replaced with a new node or sub-tree randomly. Mutation helps to introduce diversity into the population. An example of mutation is depicted in Figure 4, where the marked node is replaced with a new sub-tree.

There are proposals for other operators, e.g. *reproduction* which reproduces a new identical tree, and permutation operators which randomly permutes the values of a sub-tree (the arguments of a chosen function) [5].

In addition, there are many extensions made onto GP such as Automated Defined Functions (ADFs) [5] and Strongly Typed Genetic Programming (STGP) [11]. ADFs are "building blocks" observed during the evolution process. They can be recognised and subsequently be made available as units to the evolutionary process, effectively

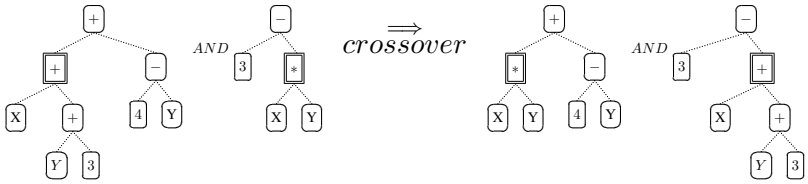


Fig. 3. Crossover in GP

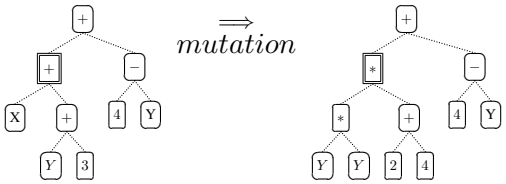


Fig. 4. Mutation in GP

implementing an on-the-fly modularisation process. On the other hand, STGP augments each node with a type and a set of type rules is defined to specify how nodes in a tree can be connected with one another. For example, we may require that $\geq$ take two floating point children and return a boolean value. The population initialisation and genetic operations must obey the type rules. In STGP only well-typed individuals are maintained in the population. For more detail in STGP, refer [11].

III. Related Work

From the security perspective, security policies are mostly written in high level forms and later followed by a series of transformations and refinements. Improvement in easing the development include automated transformation of high level human understandable rules to low level machine executable rules, automated policy conflicts and coverage checking and resolution. There are no known attempts to generating the policy automatically from previous decision examples using machine learning techniques although autonomic security policies were mention in [9] (No results have yet appeared).

On the other hand, rule inferencing techniques have been around for many years in machine learning domain. There are various approaches proposed, e.g. decision tree induction, Genetic Algorithm (GA), Genetic Programming (GP), Artificial Immune Systems (AIS), etc. In this paper, emphasis is placed on GP. In [16] a grammar-based genetic programming system called LOGENPRO (The LOGic grammar based GENetic PROgramming system) is proposed. In the performance test conducted, it is found that LOGENPRO outperforms some Induction Logic Programming (ILP) systems. In [10] a GP experiment on co-evolution between rules and fuzzy membership variables is designed. The result shows that the output set of rules and variables are well adapted to one another. In [14] an attempt is made to invent a generic rule induction algorithm using grammar based GP. The result is shown to be competitive with well known manually designed rule induction algorithms. However, in all the above mentioned cases, there have been no previous known research on rule inferencing technique in the our domain of interest — security policy.

IV. The GP Approach to Policy Inference

Most security policies can be represented as a set of `IF <condition> THEN <action>` rules. We shall attempt to discover an expression for the condition corresponding to some particular decision action (e.g. "allow read").

The leaf nodes at the bottom layer are elements of the terminal set T. At the next layer, leave nodes of the same type are joined together with operators that return *typed* values. These types may be numerical, set, vector or other user-defined types. At the next layer are the logic relational operators such as $<$ or $\in$; such an operator compares two typed values and returns a boolean value. These boolean values are combined at the higher levels with the logical composition operators such as AND, OR or NOT. The root node in a tree must evaluate to a Boolean. Strongly Typed Genetic Programming (STGP) [11] is used throughout the experiments presented in this paper.

Figure 5 shows 3 examples of well-typed individual for the experiment in inferring MLS Bell-LaPadula policy for read access that will be presented in Section V.

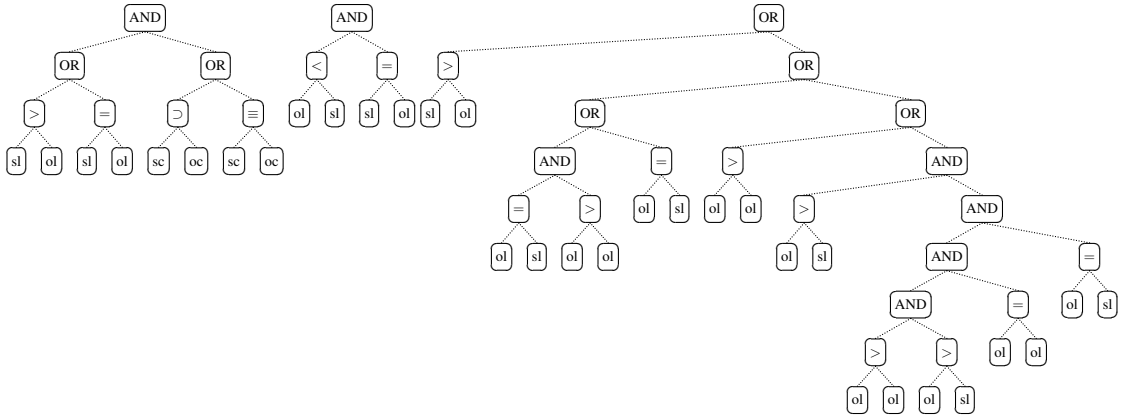


Fig. 5. Examples of well-typed individuals. The leftmost individual resembles the condition for the read access in the MLS Bell-LaPadula policy, which is $(sl > ol\ OR\ sl = ol)\ AND\ (sc \supset oc\ OR\ sc \equiv oc)$. The other two are logically equivalent individuals. They resemble only the sensitivity aspect of this policy, which is $(sl > ol\ OR\ sl = ol)$

Using this representation, the security policy inference problem can be transformed into an N-class classification problem, in which N is the number of rules in the policy. STGP is used to search for the `<condition>` part of each rule. Therefore, the number of STGP runs increases linearly with the policy size. This is not as daunting as it seems to be. As all these searches are independent from one another, this design approach can benefit from the multi-core processor revolution and execute searches in parallel. With only 1 processor, the binary decomposition method[3] can be employed to solve this problem in $N - 1$ STGP runs.

The initial population are generated randomly using the *ramp half and half* method popularised by Koza [5]. This method takes the population size, minimum and maximum heights of the trees permitted in the population as inputs and generates approximately 50% trees with maximum height while the other 50% trees have heights between the minimum and the maximum heights.

The individual fitness is computed using decision examples in a training set. Each example is represented by a vector of variables which corresponds to the decision making factors and the decision itself. The fitness of the individual is based on the number of decision examples that it agrees with. In an ideal world, it might be desirable to match all examples. However, it is often the case in practice that there are a few poor decisions and many good decisions are made. The system might be expected to evolve a policy that agrees with the majority. 100% agreement is not essential. A lower degree of agreement may simply turn the spotlight on those specific individuals with decisions inconsistent with the inferred policy. In a sense, the fitness provides a measure of how well a candidate policy agrees with examined decisions, but also acts as anomaly detector. The accumulated total score after evaluating all examples becomes the individual's fitness score. Depending on the problem, all matches (or mis-matches) are not necessarily equal. The score given for a particular match or mis-match may have profound impact on the search process and the final result. Section VI gives a good example.

After the fitness calculation stage, a new generation of individuals is produced with the use of evolutionary operators. Individuals with higher fitness scores have better chances to be selected to pass their "gene" (sub-tress) to the next generation. Further crossover and mutation operators are applied probabilistically. The evolution process continues until an individual with a "high enough" fitness score is found or a preset number of generations have elapsed.

[3]Binary decomposition method decomposes the N classes classification problem into $N - 1$ binary classification problems. The first classification problem is $(c_1, c_1{}' \equiv P - c_1)$, second problem is $(c_2, c_2{}' \equiv c_1{}' - c_2)$, $N-1$ problem is $(c_{N-1}, c_{N-1}{}' \equiv c_{N-2}{}' - c_{N-1} \equiv c_N)$. The n^{th} binary classification problem can only be solved after the $n-1$ previous problems are solved. The algorithm is inherently sequential.

V. Experiment 1: MLS Bell-LaPadula Policy Inference

In the first experiment, we concentrate on the "no read-up" part of the MLS Bell-LaPadula policy. It is simple, unambiguous and serves to demonstrate some interesting properties of our method of inference. For a read access, r, the policy can be summarised as:

$$\texttt{IF } sl \geq ol \ \ AND \ \ sc \supseteq oc \texttt{ THEN } r \texttt{ is } \textit{allowed} \tag{1}$$

$$\texttt{IF } sl < ol \ \ OR \ \ sc \not\supseteq oc \texttt{ THEN } r \texttt{ is } \textit{denied} \tag{2}$$

where sl and ol are subject and object sensitivity levels and sc and oc are subject and object category sets. Since the decisions are binary *allow/deny* decisions, the GP algorithm only needs to be run once to search for the condition for either *allow* or *deny*, the other condition can be simply obtained by logical negation. The condition for *allow* is chosen to be the learning target in this experiment.

The terminal set T consists of four variables, namely sl, ol, sc and oc but no constant value. The sl and ol are positive integers and tagged with the type "sensitivity", for which 3 operators are defined: $=$, $<$ and $>$. The $\leq$ and $\geq$ operators are intentionally omitted to make the search becomes more difficult. The sc and oc are sets and given the type "category", for which 3 operators are defined: $\equiv$, $\subset$ and $\supset$. The $\subseteq$ and $\supseteq$ operators are not included for the same reason given above. Each category in sc or oc is represented by a positive integer. The target condition, $TC(sl, ol, sc, oc)$ to be learnt in this experiment is:

$$(sl > ol \ OR \ sl = ol) \ AND \ (sc \supset oc \ OR \ sc \equiv oc) \tag{3}$$

In each run of the experiment, the maximum value of sensitivity levels for sl and ol, SNS_{max} and the total number of categories, CAT_{max} are defined. 100 randomly generated examples are used as the training set. Each example x is a vector with 5 attributes: sl_x, ol_x, sc_x, oc_x and dec_x. sl_x and ol_x are randomly chosen from $\{1 \,..\, SNS_{max}\}$; elements of sc_x and oc_x are randomly chosen from $\{1 \,..\, CAT_{max}\}$; and dec_x is set to be either 1 (*allow*) or 0 (*deny*) in accordance with the MLS Bell-LaPadula policy. Thus, all example decisions here are correct as far as MLS Bell-LaPadula policy is concerned.

The fitness of an individual (candidate policy), $fitness(i)$ is simply the sum of the matches between the decision made by the individual and the decision recorded in each example in the entire training set. Formally, let $d_{i,x}$ be the decision an individual i made for an example x; $True$ as 1; and $False$ as 0, then $fitness(i)$ is defined as in (4).

$$fitness(i) \ = \sum_{\forall \ example \ x} (d_{i,x} \equiv dec_x) \tag{4}$$

This experiment is carried out using the ECJ Framework v16 [8] and the experimental setup is summarised in Table I. For those parameters that are not specified in Table I, the default values defined in the framework are used.

Objective	Search for a TC logically equivalent condition in (3), which is $(sl > ol \ OR \ sl = ol) \ AND \ (sc \supset oc \ OR \ sc \equiv oc)$
Terminal set T	$\{sl, ol, sc, oc\}$
Functional set F	$\{AND, OR, =, >, <, \equiv, \subset, \supset\}$
Fitness function $f(i)$	$\sum_{\forall \ example \ x} (d_{i,x} \equiv dec_x)$
Number of generations	50
Population size	1024 (default)
Population initialisation	Ramp half and half method with the minimum and maximum heights of the tree set to be 2 and 6 respectively (default)
Genetic Operators (P)	Crossover (0.9), Mutation (0.1)
Maximum height of tree	17

TABLE I
Experimental setup summary of MLS Bell-LaPadula policy inference for read access

Initially SNS_{max} and CAT_{max} are set to be 5. 10 runs of the experiment, each with a training set generated with a different random seed, are carried out. In all cases, logically equivalent conditions of TC in (3) can be learnt. Then we investigated the robustness of this inference technique as follows:

- scaling up SNS_{max} and CAT_{max};
- inclusion of "wrong" examples in the training set (i.e. inconsistent decision making is demonstrated);
- changing other parameters including population size and tree height.

A. Scaling Up SNS_{max} and CAT_{max}

The same experiment is repeated using 6 different settings of (SNS_{max}, CAT_{max}): $(10, 5)$, $(20, 5)$, $(30, 5)$ and $(5, 10)$, $(5, 20)$, $(5, 30)$. In the first 3 settings, in which SNS_{max} is scaled up to 30, TC can still be found. However, in the later 3 settings, in which CAT_{max} is scaled to 30, the result changes; only weaker conditions TC' are found. More precisely, the conditions learnt using the $(5, 10)$ setting are logically equivalent to $(sc \supset oc \; OR \; sc \equiv oc)$; the conditions learnt using the $(5, 20)$ setting are either logically equivalent to $(sc \supset oc \; OR \; sc \equiv oc)$ or simply $sc \supset oc$; and the conditions learnt using the $(5, 30)$ setting are logically equivalent to $sc \supset oc$.

Randomly generated categories sets pose interesting problems from a training point of view. The probability of randomly generating a pair (sc, oc) where $sc \equiv oc$ is small, $1/(2^{CAT_{max}})$ in fact. Thus the expected number of category equality examples in a sample of size N is $N/(2^{CAT_{max}})$. Unless the system sees examples of how equality should be handled, it cannot be expected to infer how that specific condition should be handled. Inference summarises rather than speculates. As usual, the training set characteristics are important. Experiments are carried out to validate this intuition. Examples are manually created to cover the equality case and the experiment is re-run with the same setting. The results agree with this intuition; logically equivalent conditions of TC are learnt for settings with CAT_{max} equals to 10, 20 and 30.

To further investigate the effect of training set coverage, 3 experiments with extreme settings are carried out. First, a training set with 9 examples that cover all the possible combinations of $((sl, ol), (sc, oc))$ relationships in TC, namely $(>, =, <) \times (\supset, \equiv, \subset)$ is used. The learnt condition is logically equivalent to TC. At the other extreme, an experimental setup using all examples with *deny* decision ($dec_x = 0$) yields the a logically equivalent condition of $False$. Conversely, if all examples in the training set are examples with *allow* decision, then a logically equivalent condition of $True$ is learnt. Thus, a mixture of correct *allow/deny* examples is required to evolve credible policies.

B. Inclusion of "Wrong" Examples

Here the experiment is repeated with the introduction of wrong examples in the training set (i.e. so we have examples of inconsistent decision making). We first started to introduce 10% "wrong" examples and then 20%, 25% and 30%. In all cases, the conditions learnt are similar with those learnt with all correct examples. This is because the search for the condition is guided by the fitness function which is defined as the number of matches between decisions made by an individual and the ones encoded in examples. In order to have maximum fitness, the search will tend to model the correct examples (which are in the majority) and choose to be inconsistent with the others. This is encouraging because 100% agreement is not the actual goal as mentioned earlier. Highlighting anomalous behaviours is also important.

C. Parameter Changes

Each experiment described so far is repeated using training set size consists of 500 and 1000 randomly generated examples. The conditions learnt in each experiment are very similar with the conditions learnt using only 100 examples.

Also, each experiment is repeated with various population sizes: 50, 100, 500 and 5000. When the population size is 500 or larger, logically equivalent conditions of TC are learnt, and there is no significant difference in terms of the number of generations required. However, the execution time per generation does increase significantly due to the increase in the amount of genetic operations performed. When the population size is set to be 50 and 100, the desired condition cannot be learnt sometimes. Investigation is made on these populations using the GUI provided by ECJ. Diversity in the population is lost in early generations, i.e. premature convergence in the population occurs.

To investigate the effect of tree size, the experiment is repeated with different maximum tree heights. In each experiment, the maximum tree height is set to be one less than that of the previous experiment; the first experiment has maximum tree height of 17. The target condition can not be learnt if the tree height is less than 4, which is the minimum height to represent TC. The results also show that the number of generations needed to learn the condition increases as the maximum tree height used increases, because larger tree height implies larger search space.

VI. Experiment 2: Fuzzy MLS Policy Inference

Up to this point, we have only considered binary decision making. In this experiment, we concentrate on learning the conditions of a risk-based policy that could have more than two decisions, beyond the *allow* and *deny* binary decision model. The policy model is the Fuzzy MLS model [2], [3]. This policy uses the risk-based rationale of the MLS Bell-LaPadula policy to compute a *quantified risk estimate* by quantifying the "gap" between a subject's label and an object's label in an MLS system.

Quantified risk estimates are numbers and therefore could be used to build the risk scale shown in Figure 6 [2]. The risk scale is divided into multiple bands. Each band is associated with a decision. The risk in the bottom band

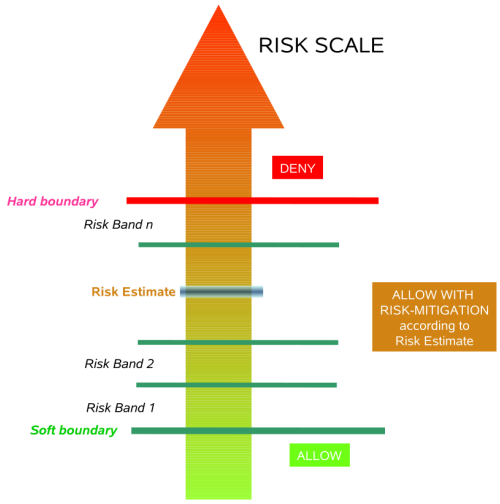


Fig. 6. Risk adaptive access control on a risk scale

is considered low enough so the decision is simply *allow* whereas the risk in the top band is considered too high so the decision is *deny*. Each band between the top and bottom is associated with a decision *allow with different certain risk mitigation measures*.

The Fuzzy MLS model defines risk as the *expected value of damage* caused by unauthorised disclosure of information:

$$risk = (value\ of\ damage) \times (probability\ of\ unauthorised\ disclosure) \tag{5}$$

The value of damage is estimated from the object's sensitivity level. The probability of unauthorised disclosure is estimated by quantifying two "gaps": one between the subject's and the object's sensitivity levels and the other between the subject's and the object's category sets. For simplicity, this experiment looks only at the sensitivity levels and assumes the categories sets are the same[4], and thus risk becomes a function of subject's and object's sensitivity levels only. For more detail on risk quantification, refer [2], [3].

Since there is no discussion on the way to partition the scale into risk bands in [2], the following formula is defined to map a risk number to a risk band:

$$band(risk(sl, ol)) \ = \ min(\lfloor log_{10}(risk(sl, ol)) \rfloor, N) \tag{6}$$

[4]Therefore the gap between categories sets is 0.

where $(N+1)$ is the number of bands desired on the scale and the function $risk(sl, ol)$ is defined in Appendix A according to [2]. Base-10 logarithm is used in (6) to compute the order of magnitude of risk as the band number. In our experiments, the scale is divided into 10 bands ($N = 9$) numbered from 0 to 9. Therefore, 10 STGP runs are required to search for conditions for all the bands. This experiment serves as a good illustration of the N-classes classification problem discussed in Section IV such that each band corresponds to a class.

Since only the sensitivity levels are considered, the terminal set T has only two variables, namely sl and ol. However, an additional set of real constant number in the range of $(-1, 1)$ is added to T to satisfy the sufficiency property. This is because the target condition, TC can no longer be represented with only sl and ol. Elements in T are given type *sensitivity*, for which 15 operators are defined: 3 relational operators ($=$, $<$ and $>$) which returns a value of type "boolean" and 12 arithmetic operators $(+, -, *, /, exp, log, sin, cos, max, min, ceil, floor)$ which returns a value of type "sensitivity". The target condition for band j, TC_j to be learnt is:

$$TC_j(sl, ol) \;=\; (band(risk(sl, ol)) \;\equiv\; j) \tag{7}$$

In other words, STGP is used to search for an equivalent function of the composition of 2 functions: the band function and risk function.

For all experiments described in this section, SNS_{max} is set to be 10 and an example x in a training set is a triple $(sl_x, ol_x, band_x)$, where $band_x = band(risk(sl_x, ol_x))$. In the first experiment, we generated a training set with $10 \times 10 = 100$ examples that cover all the possible (sl, ol) pairs and see what can be learnt in this optimal setting. The numbers of examples in each band is shown in Table II. Band 0 has the most number of examples because it includes all the low-risk cases where $sl > ol$. All the other bands have relatively small number of examples.

Band	Number of examples
0	52
1	5
2	3
3	4
4.	3
5	5
6	5
7	7
8	8
9	8

TABLE II
DISTRIBUTION OF EXAMPLES IN THE TRAINING SET

As in the MLS Bell-LaPadula Experiment in Section V, the fitness score for an individual in the search for condition of band j, $fitness_j(i)$ is defined to be the total number of correct decisions i made:

$$fitness_j(i) = \sum_{\{x|band_x \equiv j\}} (d_{i,x} \equiv True) \;+\; \sum_{\{x|band_x \not\equiv j\}} (d_{i,x} \equiv False) \tag{8}$$

The experimental setup is summarised in Table III. As before, this experiment is carried out using the ECJ Framework v16 [8] and the default values are used for the unmentioned parameters.

For each band, 10 runs are conducted. However, the result shows that only the target condition for band 0, TC_0 can be learnt.

Investigations on the experimental logs for other bands reveals the following facts:

- the fittest individuals remain unchanged after a few generations.
- individual fitness scores converge quickly to that of the fittest individual. This suggests that the searches for band in $\{1 \;..\; 9\}$ quickly become random searches because equivalent fitness implies equivalent probability one individual to be selected.
- for a band in $\{1 \;..\; 9\}$, the fitness score of the fittest individual is $100 - (number\ of\ examples\ in\ the\ band)$. This suggests that all points in fitness score come from negative examples (example not in that band in

Objective	Search for condition that classifies all examples into a risk band j accurately, which is, $\forall x,\ band(risk(sl_x, ol_x)) \equiv band_x$
Terminal set T	$\{sl, ol\} \cup \{r \mid -1.0 < r < 1.0\} \cup \{TRUE, FALSE\}$
Function set F	$\{+, -, *, /, exp, log, sin, cos, max, min, ceil, floor\} \cup \{AND, OR, =, >, <\}$
Fitness function $f_j(i)$	$\sum_{\{x \mid band_x \equiv j\}} (d_{i,x} \equiv True) + \sum_{\{x \mid band_x \not\equiv j\}} (d_{i,x} \equiv False)$
Number of generations	500
Population size	1024 (default)
Population initialisation	Ramp half and half method with the minimum and maximum heights of the tree set to be 2 and 6 respectively (default)
Genetic Operators (P)	Crossover (0.9), Reproduction (0.1)
Maximum height of tree	17

TABLE III
EXPERIMENTAL SETUP SUMMARY OF FUZZY MLS POLICY INFERENCE FOR READ ACCESS USING IF−THEN APPROACH

question), implying these examples completely outweigh the relative few examples in that band. This also explains why TC_0 can be learnt since more than half of the examples are in band 0.

A. Weight and Punishment on Fitness Score

To improve the result, we assign different weights to the fitness scores of different kinds of decisions made by an individual according the following principles. In the search for condition of band j, TC_j:

- For a correct decision, *award more* if
  - the risk is higher for security concerns; i.e., award more for a larger j.
  - the decision is *is true positive* (hits the target); i.e., award more when $band_x \equiv j$. This will overcome the effect that relative few positive examples are in band j when $j \neq 0$.
- For an incorrect decision, *punish more* if
  - the decision is *false positive* ($d_{i,x} \equiv True$ and $band_x \neq j$) and is *more off the target*; i.e., punish more as $\mid j - band_x \mid$ becomes larger. Also, for security concerns, punish more if this false positive decision *underestimates the risk*; i.e., punish more if $band_x > j$.
  - the decision is *false negative* ($d_{i,x} \equiv False$ and $band_x = j$) when the risk is higher for security concerns; i.e., punish more for a larger j.

Using these principles, the fitness function is changed to be:

$$\begin{aligned} fitness_j(i) \;=\; & \sum_{\{x \mid band_x \equiv j\}} w_{tp}\{d_{i,x} \equiv True\} \;+\; \sum_{\{x \mid band_x \not\equiv j\}} w_{tn}\{d_{i,x} \equiv False\} \;- \\ & \sum_{\{x \mid band_x \not\equiv j\}} w_{fp}\{d_{i,x} \equiv True\} \;-\; \sum_{\{x \mid band_x \equiv j\}} w_{fn}\{d_{i,x} \equiv False\} \end{aligned} \tag{9}$$

where

$$\begin{aligned} w_{tp} &= j + 1, \\ w_{tn} &= (j+1)/10, \\ w_{fp} &= \begin{cases} band_x - j & \text{if } band_x > j, \\ (j - band_x)/2 & \text{if } band_x < j, \end{cases} \\ w_{fn} &= j + 1 \end{aligned}$$

Experiment is repeated 3 times, each with a different training sets. The first training set consists of all possible 100 (sl, ol) pairs where sl and ol are integers in $[0, 9]$. This optimal setting lends itself to act as the control. The second and third sets consist of 100 and 500 randomly generated (sl, ol) pairs where sl and ol are also integers in $[0, 9]$. Unlike in the control, these two sets would have incomplete coverage and uneven distribution of examples over risk bands.

After going through the evolution process, the best individual in the population is selected to test against two sets of examples. The first set is same as the first 100-example training set. This testing set provides a good indication on how much "knowledge" has been acquired by the approach employed in a fixed number of generations. The second testing set consists of 100 randomly generated (sl, ol) pairs where sl and ol are *real numbers* in $[0.0, 9.0]$. Therefore, most of these examples are unseen yet similar to training examples. This set provides a good measure on how much the acquired knowledge can be applied for unseen cases. The average performance of the policies over 10 runs are summarised in the Table IV.

Experiment	Training set	Testing set	Distance from target band				Mean distance
			0	1	2	≥ 3	
IF-THEN Rules	1	1	88.1	0.4	0.7	10.8	0.797
		2	48.1	19.0	5.5	27.4	1.831
	2	1	82.7	3.7	2.3	11.3	0.746
		2	52.7	20.9	5.4	21.0	1.500
	3	1	93.8	0.4	0.0	5.8	0.348
		2	49.6	20.9	4.5	25.0	1.824

TABLE IV
PERFORMANCE SUMMARY OF THE LEARNT POLICIES USING IF-THEN RULES APPROACH

The policies learnt perform extremely well in Testing Set 1 with accuracy of about 88.2% of the examples are mapped correctly and average of 0.63 band distance from the target band. However, the performance of the policies on unseen Testing Set 2 decreases significantly to only 50.1% of the examples are mapped correctly with average of 1.72 band distance from the target band. We suspect this problem is due to the inherent problem in binary making decision process. We will see in next section in which this problem can be alleviated.

VII. FUZZY SET CLASSIFICATION

In practice, a diverse training set may not be available, especially when examples are not provided all at once but gradually over time. We propose an approach based on a *modified fuzzy set concept* to address this issue. Fuzzy logic concept was conceived by Lotfi Zadeh to deal with this ambiguity in [17]. Since its introduction, it has been used in various domains to do decision/knowledge inference especially in the circumstance with uncertainty. For detail introduction on fuzzy set, refer [7].

Considering how human beings learn, a person usually will not be able to hit the target on the first attempt. Instead, it is more likely his performance would gradually improve over time; i.e., *the distance* to the target is *gradually shortened*. Along the learning process, knowledge on those which are not the target, such as their distances to the target, is acquired. We argue that the knowledge about the target and the non-targets is all useful in addressing the issue mentioned above.

In both experiments V and VI, each decision is treated as single discrete set. The use of distance knowledge is implicitly encoded in the fitness function. In Experiment V, the score of 1 is awarded to an individual fitness if the individual's decision is on target (matches decision in the example) and 0 otherwise. In Experiment VI, the knowledge of "distances" is especially useful. The weighted fitness function, $fitness_j(i)$ in (9) is essentially using a distance measure, namely the difference in risk band numbers, to calculate the score.

In this proposal, each class is treated as a fuzzy set and the distance knowledge is used to compute fuzzy set memberships. The learning targets are the fuzzy set membership functions for the classes. Here, the emphasis is placed on solving N-classes classification problem where a total order relationship exists among the classes. The risk bands classification problem in a Fuzzy MLS policy discussed in section VI is a good example and will be used to illustrate this proposal.

To guide the learning of the fuzzy set membership function for band j, $M_j(sl, ol)$, the *target membership* of each of the 10 bands in the band j fuzzy set is first defined. Essentially, these 10 pre-defined points characterise the shape and location of the M_j curve. The learning process becomes a *curve fitting* exercise to search for a curve that best fits the 10 points, using all the examples in the training set. Curve fitting is naturally more tolerant

of incomplete coverage in the training set because it uses *interpolation* and *extrapolation* to compensate for the "missing points". It is also more resilient to a few out-liars in the training set.

Furthermore, the fuzzy membership range is changed to $[-1.0, 1.0]$ with 0.0 represents full membership. This is different from the traditional fuzzy membership range, $[0.0, 1.0]$ with 1.0 representing full membership. The expansion on the negative range allows the information about the direction (left or right to the target band) to be encoded. For example, for $M_5(sl, ol)$, the target membership of each band, starting from band 0, can be defined as:

$$[-0.5, -0.4, -0.3, -0.2, -0.1,\ 0.0,\ 0.1,\ 0.2,\ 0.3,\ 0.4]$$

Band 5 has membership 0. The target memberships for other $M_{j\neq 5}$ can be defined similarly. With all 10 membership functions learnt, one can determine the band of an input by feeding the input to all 10 functions and computing the band number by examining the 10 membership values returned by these functions. For example, if each membership value indicating that "the distance between the input and my band places the input close to band 5", then with very high confidence we can say the input belongs to band 5. This is analogous to examine the input from 10 different perspectives to draw the final conclusion, which is more likely to be accurate than examining the input from one perspective.

Two setups with different pre-defined target memberships are carried out to validate this concept. In the first setup the 10 target memberships for M_j are defined as :

$$M_j(k) \ \equiv \ (k-j)/10,\ k = 0\ to\ 9 \tag{10}$$

This is like mapping a traditional triangle fuzzy membership function, which has the range $[0, 1]$ and $M_j(j) \equiv 1$ as the tip of the triangle, to a straight-line membership function with the range $[-1, 1]$ and $M_j(j) \equiv 0$. Figure 7 shows the target membership curves for all 10 bands using (10). In the second setup, bell-shaped Gaussian distribution curves are mapped in a similar fashion and Figure 8 shows the membership curves for all 10 bands.

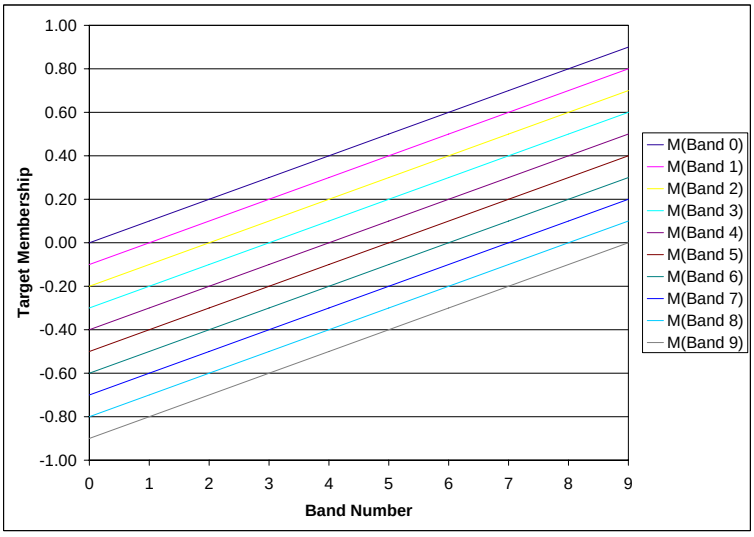


Fig. 7. The linear (modified triangle) membership functions, M_{linear} for of all the bands

The terminal set T consists of sl, ol and a set of random real constant numbers in the range of $(-1, 1)$ whereas the function set F is reduced to $\{+, -, *, /, exp, log, sin, cos\ max, min, ceil, floor\}$ (i.e the comparators and boolean operators are removed).

The fitness function uses per-band *normalised* distance: let $M_{j,i}$ represent an individual i in the search for the membership function for band j, the individual fitness, $fitness_j(i)$ is defined as follow:

$$fitness_j(i) = \sum_{\forall\ bands\ k=0\ to\ 9} \frac{\sum_{\forall\ x\ in\ band\ k} |M_{j,i}(sl_x, ol_x) - M_j(band_k)|}{total\ number\ of\ band\ k\ examples} \tag{11}$$

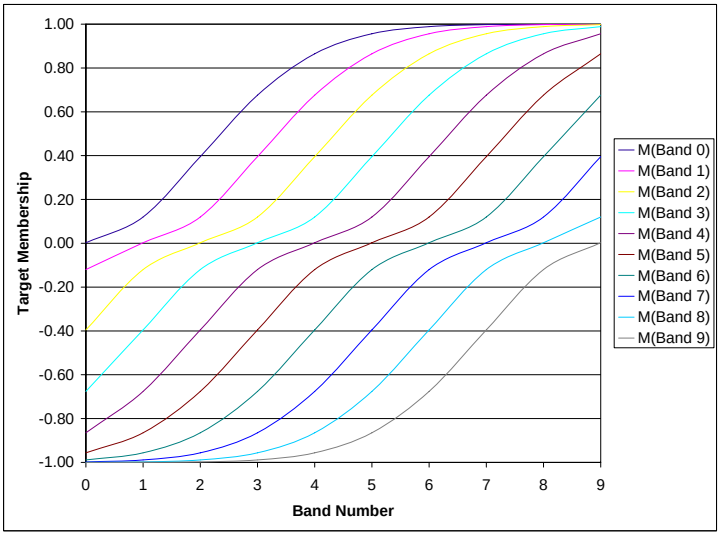


Fig. 8. The curve (modified Gaussian distribution) membership functions, $M_{Gaussian}$ for of all the bands

Thus, the $fitness_j(i)$ represents the sum of *normalised* distances between the $M_{j,i}$ and M_j. As the fitness score represents the distance between the $M_{j,i}$ and M_j; and in this case a smaller score means better fit.

Once all 10 membership functions are learnt and feeding an input $x \equiv (sl_x, ol_x)$ to these functions, we need a *de-fuzzification* mechanism to map all 10 values returned by these functions to a risk band number. Two voting based algorithms are used for de-fuzzification: One (Algorithm 1) uses the direction knowledge only and the other one (Algorithm 2) uses both the direction and distance knowledge.

initialise an array $v[10]$ *with all elements set to* 0
forall *example* x **do**
  forall *band* j **do**
    if $M_j(sl_x, ol_x) > 0.05$ **then**
      forall $k > j$ **do**
        $v[k] \leftarrow v[k] + 1$
    else if $M_j(sl_x, ol_x) < -0.05$ **then**
      forall $k < j$ **do**
        $v[k] \leftarrow v[k] + 1$
    else if $-0.05 \leq M_j(sl_x, ol_x) \leq 0.05$ **then**
      $v[j] \leftarrow [j] + 1$
choose $v[i]$ *with the maximum value*
outputs i *as the risk band number*

Algorithm 1: Direction based de-fuzzification

The experiment setup is summarised in Table V. This experiment is carried out using the ECJ Framework v16 [8] and the default values are used for unmentioned parameters.

As before, 10 runs of the experiment using each of the 3 training sets are carried out and the best policy evolved in each run is evaluated against on the same 2 testing sets. The average performance of the policies over 10 runs are summarised in the Table VI.

In Testing Set 1, using only direction knowledge, the number of examples mapped correctly with the policies learnt decreases to only 53.6% of the examples for the mapped triangle fuzzy membership approach and 64.1%

initialise an array $v[10]$ *with all elements set to* 0
forall *example* x **do**
  forall *band* j **do**
    if $M_j(sl_x, ol_x) > 0.05$ **then**
      $p \leftarrow j + M_j(x) * 10$
      $k \leftarrow min(\lfloor p + 0.5 \rfloor, 9)$
      $v[k] \leftarrow v[k] + 3$
      if $k > p$ **then**
        $v[k-1] \leftarrow v[k-1] + 2$
        $v[k+1] \leftarrow v[k+1] + 1$
      else
        $v[k-1] \leftarrow v[k-1] + 1$
        $v[k+1] \leftarrow v[k+1] + 2$
    else if $M_j(sl_x, ol_x) < -0.05$ **then**
      $p \leftarrow j + M_j(x) * 10$
      $k \leftarrow max(\lceil p - 0.5 \rceil, 0)$
      $v[k] \leftarrow v[k] + 3$
      if $k < p$ **then**
        $v[k+1] \leftarrow v[k+1] + 2$
        $v[k-1] \leftarrow v[k-1] + 1$
      else
        $v[k+1] \leftarrow v[k+1] + 1$
        $v[k-1] \leftarrow v[k-1] + 2$
    else if $-0.05 \leq M_j(sl_x, ol_x) \leq 0.05$ **then**
      $v[j] \leftarrow v[j] + 3$
      $v[j+1] \leftarrow v[j+1] + 1$
      $v[j-1] \leftarrow v[j-1] + 1$
choose $v[i]$ *with the maximum value*
outputs i *as the risk band number*

Algorithm 2: Direction and distance based de-fuzzification

Objective	Search for the fuzzy membership functions for all bands, $\forall j \in [0, 9], M_j$ characterised by the examples in the training set
Terminal set T	$\{sl, ol\} \cup \{r \mid -1.0 < r < 1.0\}$
Function set F	$\{+, -, *, /, exp, log, sin, cos\ max, min, ceil, floor\}$
Fitness function $f_j(i)$	$fitness_j(i)$ in (11)
Number of generation	500
Population size	1024 (default)
Population initialisation	Ramp half and half method with the minimum and maximum heights of the tree set to be 2 and 6 respectively (default)
Genetic Operators (P)	Crossover (0.9), Reproduction (0.1)
Maximum height of tree	17

TABLE V
EXPERIMENTAL SETUP SUMMARY OF FUZZY MLS POLICY INFERENCE FOR READ ACCESS USING FUZZY MEMBERSHIP APPROACHES

Experiment	Training set	Testing set	Distance from target band				Mean distance
			0	1	2	≥ 3	
Mapped triangle (Direction only)	1	1	50.7	26.4	11.5	11.4	0.868
		2	40.5	33.0	13.1	13.4	1.076
	2	1	56.3	23.9	9.2	10.6	0.826
		2	45.0	31.7	8.6	14.7	1.056
	3	1	53.9	23.5	11.1	11.5	0.863
		2	43.4	36.6	10.9	9.1	0.921
Mapped Gaussian curve (Direction only)	1	1	63.9	20.6	8.3	7.2	0.620
		2	51.8	29.7	12.7	5.8	0.757
	2	1	56.9	20.9	14.4	7.8	0.772
		2	41.8	25.4	18.9	13.9	1.173
	3	1	71.4	19.4	6.7	2.5	0.414
		2	60.1	24.2	9.2	6.5	0.638
Mapped triangle	1	1	91.6	4.8	1.0	2.6	0.178
		2	62.2	23.5	3.6	10.7	0.781
	2	1	76.0	13.3	3 .6	7.1	0.514
		2	62.1	23.9	4.8	9.2	0.714
	3	1	78.5	10.8	2.5	8.2	0.535
		2	58.8	27.3	3.4	10.5	0.782
Mapped Gaussian curve	1	1	65.5	14.4	9.6	10.5	0.696
		2	52.6	18.8	12.7	15.9	1.013
	2	1	62.6	14.6	9.2	13.6	0.916
		2	47.9	17.6	11.9	22.6	1.492
	3	1	65.4	14.5	9.0	11.1	0.707
		2	50.3	19.7	12.3	17.7	1.097

TABLE VI
PERFORMANCE SUMMARY OF THE LEARNT POLICIES USING FUZZY MEMBERSHIP APPROACHES

of the examples for the mapped Gaussian fuzzy membership approach. Having said that, the performance in terms of the mean band differences remain similar as before with 0.85 band difference for the mapped triangle fuzzy membership approach and 0.60 for the mapped Gaussian fuzzy membership approach. With the use of additional distance knowledge, the number of examples mapped correctly with the policies learnt using the mapped triangle membership approach increases to 82.0% of the examples with mean band difference of 0.41. However, the number of examples mapped correctly with the policies learnt using the mapped Gaussian approach remains similar as before, 64.5% of the examples with mean band difference of 0.77.

More importantly, the performance of the policies learnt using fuzzy membership functions on Testing Set 2 show some significant improvements. Using only direction knowledge, the policies learnt using mapped triangle and mapped Gaussian fuzzy membership approaches map 43.0% and 51.2% of the examples to the correct bands with the mean of band differences remain as low as 1.02 and 0.86 band differences respectively. These are significantly lower than the 1.72 band difference using the `IF-THEN` approach. With the use of additional distance knowledge, the number of examples mapped correctly becomes 61.0% of the examples with mean band difference of 0.73 for the mapped triangle fuzzy membership approach whereas the number of examples mapped correctly remains at 50.3% of the examples with the mean band difference degraded to 1.20 for the mapped Gaussian approach.

Overall, the experimental result does suggest that it is feasible for the fuzzy membership based approach to work well on both testing sets without a diverse training set although issues like target membership assignment and fitness evaluation still requires much more research. Also, the de-fuzzification method that uses both direction and distance knowledge improves the performance of the experiments using the mapped triangle based fuzzification, the mean distance from the target for all cases is reduced to less than 0.8. However, this new de-fuzzification method degrades the performance of the experiments using the mapped Gaussian curve based fuzzification. This is because the de-fuzzification method makes the assumption that the distance from the target increases linearly in respect to the membership value. This is not the case in Gaussian curve based fuzzification. A possible further work is to design a compatible de-fuzzification mechanism using both distance and direction for Gaussian curve fuzzification.

VIII. Conclusion and Further Work

We proposed a generic policy inference framework using the classification approach, implemented using Genetic Programming. Two experiments are conducted to validate the proposal. The results show that the learning process is largely independent of many parameters, but the training set and fitness functions do play crucial roles. A training set has to be sufficiently diverse. This concurs with our intuitions that a machine cannot learn something that it has not seen before. To address this issue, a new fuzzy set based approach is proposed and experiments are conducted to test its feasibility. The result shows that such fuzzy models can achieve comparable accuracy with their corresponding classic models yet depends less on the diversity of training sets, but more research is needed on defining target fuzzy memberships and fitness evaluation function.

Also, the results indicate that the distribution of examples in the training set also matters. The fitness function can lost its guiding feature in the absence of evenly distributed examples. Weights may be introduced into fitness functions to "flatten" the distribution. However, it is not always possible to define appropriate weights. A radical way to put this matter forward may be to employ multi-objectives genetic programming (MOGP).

Acknowledgment

This research is funded as part of the International Technology Alliance (ITA) Project 6: Risk and Trust Management in Dynamic Coalition. We would like to thank Aaron Kershenbaum, Dakshi Agrawal of IBM Watson T. J. Watson Research Centre, Hawthorne in providing valuable feedback and facilities to conduct this research. This research is funded as part of the International Technology Alliace (ITA) Project 6: Risk and Trust Management in Dynamic Coalition.

References

[1] D. Bell and L. LaPadula. Secure computer systems: Mathematical foundations. Technical Report ESD-TR-73-278, MITRE Corporation, 1973.

[2] Pau-Chen Cheng, Pankaj Rohatgi, Claudia Keser, Paul A. Karger, Grant M. Wagner, and Angela Schuett Reninger. Fuzzy Multi-Level Security: An Experiment on Quantified Risk-Adaptive Access Control. Technical report, IBM Research Report RC24190, 2007.

[3] Pau-Chen Cheng, Pankaj Rohatgi, Claudia Keser, Paul A. Karger, Grant M. Wagner, and Angela Schuett Reninger. Fuzzy Multi-Level Security: An Experiment on Quantified Risk-Adaptive Access Control. *IEEE Symposium on Security and Privacy*, pages 222–230, 2007.

[4] John R. Koza. Hierarchical genetic algorithms operating on populations of computer programs. In *IJCAI*, pages 768–774, 1989.

[5] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[6] John R. Koza, David Andre, Forrest H. Bennett, and Martin A. Keane. *Genetic Programming III: Darwinian Invention & Problem Solving*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[7] Yow Tzu Lim and John Andrew Clark. Ph.D. qualifying thesis: Survey on risk-based decision making. Technical report, University of York, York, United Kingdom, July 2007.

[8] Sean Luke. ECJ version 16 A Java-based evolutionary computation research system, August 2007.

[9] Patrick Drew McDaniel. Policy evolution: Autonomic environmental security, December 2004.

[10] Roberto R. F. Mendes, Fabricio de B. Voznika, Julio C. Nievola, and Alex A. Freitas. Discovering fuzzy classification rules with genetic programming and co-evolution. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, page 183, San Francisco, California, USA, 7-11 2001. Morgan Kaufmann.

[11] David J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.

[12] The MITRE Corporation JASON Program Office. Horizontal integration: Broader access models for realizing information dominance. Technical Report JSR-04-132, The MITRE Corporation JASON Program Office, Mclean, Virginia, Dec 2004.

[13] Michael O'Neill and Conor Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language*, volume 4 of *Genetic programming*. Kluwer Academic Publishers, 2003.

[14] GL Pappa and AA Freitas. Towards a genetic programming algorithm for automatically evolving rule induction algorithms. In J. Furnkranz, editor, *Proc. ECML/PKDD-2004 Workshop on Advances in Inductive Learning*, pages 93–108, Pisa, Italy, September 2004.

[15] Hartmut Pohlheim. Geatbx: Genetic and evolutionary algorithm toolbox for use with matlab, 2005.

[16] Man Leung Wong and Kwong Sak Leung. *Data Mining Using Grammar Based Genetic Programming and Applications*, volume 3 of *Genetic Programming*. Kluwer Academic Publishers, January 2000.

[17] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, June 1965.

APPENDIX

In [2], the risk resulted from the "gap" between a subject's and an object's sensitivity levels (sl and ol) is estimated using the following formula:

$$risk(sl, ol) \;=\; Val(ol) \times P_1(sl, ol) \tag{12}$$

$Val(ol)$ is the estimate value of damage and is define in [2] as

$$Val(ol) = a^{ol}, a > 1$$

The object sensitivity level is considered to be the *order of magnitude* of damage and hence $Val(ol)$ is defined as an exponential formula. In our experiments we set a to be 10. $P_1(sl, ol)$ is the probability of unauthorised disclosure and is defined in [2] as a sigmoid function:

$$P_1(sl, ol) = \frac{1}{1 \;+\; exp(-k(TI(sl, ol) - mid))}$$

$TI(sl, ol)$ is called the *temptation index* which indicates how much the subject with sensitivity sl is tempted to leak information with sensitivity level ol; it is defined as:

$$TI(sl, ol) = \frac{a^{(ol-sl)}}{M - ol}$$

The intuition for $P_1(sl, ol)$ and $TI(sl, ol)$ can be found in [2]. The value mid is the value of TI that makes P_1 equal 0.5; the value k controls the slope of P_1. The value M is the *ultimate object sensitivity* and the temptation TI approaches infinity as ol approaches M; the intuition is access to an object with sensitive level equals to or more than M should be controlled by human beings and not machines. In our experiments, the maximum value (SNS_{max}) for sl and ol is 10; the settings for k, mid and M are $k = 3,\ mid = 4,\ M = 11$.