

IBM Research Report

Dynamic Security Policy Learning

Yow Tzu Lim¹, Pau Chen Cheng², Pankaj Rohatgi², John A Clark¹

¹Department of Computer Science
University of York
UK

²IBM 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

Dynamic Security Policy Learning

Yow Tzu Lim*, Pau Chen Cheng[†], Pankaj Rohatgi[†], John A Clark*

*Department of Computer Science, University of York, UK.

Email: {yowtzu, jac@cs.york.ac.uk}

[†]IBM Watson Research Center, USA.

Email: {pau, rohatgi@us.ibm.com}

Abstract—Recent research [1,2] has suggested traditional top down security policy models were too rigid to cope with changes in dynamic operational environments. There is a need for more flexibility in security policy to protect the information and yet still satisfy the operational needs. Previous work has shown that a security policy can be learnt from examples using machine learning techniques. Given a set of criteria of concern, one can apply these techniques to learn the policy that best fits the criteria. These criteria can be expressed in terms of high level objectives, or characterized by the set of previously seen decision examples. We argue here that even if an optimal policy could be learnt automatically, it will eventually become sub-optimal over time as the operational environment changes. In other words, the policy needs to be continually updated to maintain its optimality. In this paper, we review the requirements for dynamic learning and propose a dynamic policy learning framework.

Index Terms—Dynamic Learning, Concept Drift, Multi Objective Genetic Programming, MOGP

I. INTRODUCTION

A traditional top-down, rigid security policy model defines a static trade-off between the estimated risk and benefit that can be obtained from allowing accessing the information in the system. It is often difficult to develop a policy that encodes the appropriate risk vs. benefit trade-off. First, there are too many factors to be considered, and some of these factors can conflict with one another. Second, due to the nature of the information — exposure of information is irreversible, the policy developed is often pessimistic, i.e. information access is denied by default to minimise the risk although it is not necessarily the best decision. Worse, exceptions to policies and additional privileges are often granted to users to meet operational needs and these privileges are rarely revoked. Conversely, sometimes information is also classified at a lower sensitivity level to facilitate information sharing. In summary, the crafted policy is sub-optimal and is tweaked during operational time in an unmanaged way to fit the operational needs.

Recent work has shown that machine learning techniques (see Section II) can be used to learn policy. Given a set of criteria of concern, one can apply these techniques to learn the policy that best fits the criteria. These criteria can be expressed in terms of high level objectives, or characterised by the set of previously seen decision examples. We argue here that even if an optimal policy could be learnt automatically, it will eventually become sub-optimal over time as the operational environment changes. In other words, the policy needs to be continually updated to maintain its optimality, especially in a

highly dynamic operational environment. There is a need for such dynamic learning to fully meet the operational criteria. In this paper, we review the requirements in dynamic learning and propose a dynamic policy learning framework.

We choose to use Genetic Programming [3], a kind of evolutionary algorithm that was previously shown to be able to learn the policy given a set of decision examples [4]–[6]. The main reason to use Genetic Programming is its rather weak assumptions and its ability to search for solutions of unknown (or controlled) size and shape in vast, discontinuous solution spaces. Other data mining algorithms are potentially applicable.

The main contributions in this paper includes:

- 1) The design of a dynamic security policy model. Currently there are no decision examples that are available for us to work with, we need to design this model and generate examples to be used for training purposes.
- 2) The discovery of the problems and difficulties encountered in dynamic learning using Genetic Programming.
- 3) The design of two dynamic learning frameworks using Multi Objective Genetic Programming (MOGP). Both of them are sufficient generic to be used with other data mining algorithms.

The rest of the paper is organised follows: Section II discuss the related work on security policy learning. Section III presents a time-varying risk budget based policy. Section IV presents the learning technique in use. Section V presents the general experimental setup. Sections VI and VII presents the experiments carried out on static and dynamic learning respectively. Section VIII presents the analysis on various way in selecting the best solution from a set of candidate solutions generated by MOGP. Finally Section IX concludes and discusses future research.

II. RELATED WORK

In this section we will first review prior work on applying machine learning techniques to learn security policies, and then we will briefly review some data stream classification algorithms that provided the inspiration for the development of our policy learning framework. See [7,8] for details on classification algorithms in data stream mining.

A. Application of Machine Learning Techniques in Policy Learning

Machine learning techniques have achieved significant success in intrusion detection systems (IDS) to discover useful patterns and rules in terms of system normal behaviour or attack features. These extracted rules are then used to recognise anomalies and intrusions. Most work focuses on using various data mining techniques to learn a classifier or clusterer that describes the normal (or abnormal) conditions of the systems. For details, refer to [9].

In [10] autonomic security policies were mentioned yet no known results have been published. In [11]–[13], various data mining techniques have been used to aid the process of role engineering: deriving roles and their associated permission from existing data. In [14] machine learning techniques are used to generalise low-level rules to high level rules. Genetic Programming [4]–[6] and later Grammatical Evolution [15] are used to learn a new risk based security policy — Fuzzy MLS [16] from a set of decision examples. Both results show these approaches are promising.

Sometimes, it is necessary to have specific policies on a per-application basis. In [17], Inoue et al. inferred a sand-boxing policy for specific application written in Java by monitoring program execution. In [18], Scott et al. introduced SPECTRE, a tool to secure a specific web application. When using SPECTRE in an inference mode, it is able to learn automatically the policy for a security gateway in Security Policy Description Language (SPDL) by monitoring the interaction between web application and its client.

B. Data Stream Classification and Dynamic Learning

Data stream classification posts two challenges to the traditional data classification algorithms: infinite data flow and concept drifts. Infinite data flow prohibits a classification algorithm to have multiple scans of the data set. Furthermore, the distribution of the data changes over. This change is typically referred to as *concept drift*. The algorithms have to be able to cope with this.

To do this, traditional algorithms have been revised to include fading effects for the older examples. A previously learnt classifier is required to undergo revision and to relearn the new concept constantly. Using the decision tree classifier as an example, the decision tree/sub-tree is pruned, re-grown or discarded as necessary [19]. The resulting algorithms are often complicated. Worse, as these algorithms discard old examples at a fixed rate, the learnt classifier is only supported by the latest data. This usually results in large prediction variances.

The ensemble of classifiers is another approach that has become very popular in data stream classification. This approach offers several advantages over single model classifier. First, it offers an efficient way to improve accuracy. Second, its parallel nature is easy to scale. Often, the ensembles approach splits the learning process into 2 parts: data summarisation and model selection. Data in the stream is divided into chunks; a classifier ensemble (which itself can also be a set of classifier ensembles) is built from each data chunk. These classifier

ensembles are combined using different weights to form the ultimate classifier. The criteria used for selecting the weights include the time (sliding window), *estimated* accuracy using the latest data chunk, variation in class distribution, etc.

Dynamic Policy Learning is very similar to data stream classification. Each possible decision in the policy can be viewed as a class; the learning objective is to search for the classifier that best agrees with the examples in the data stream in a timely manner. In both cases, the amount of data will inevitably increase over, the algorithm has to be able to learn incrementally and cope with change. Yet there is still a small distinction in terms of the learning rate requirement. In dynamic policy learning, the learning time requirement is much relaxed. A few minutes to hours for learning is acceptable. This implies we do not have the one pass learning constraint in data stream mining and old data can be revisited if necessary.

One possible approach to this problem is to apply static learning algorithm on the latest data. In other words, the old data is discarded indiscriminately after a fixed time period. Whilst this approach is conceptually simple, it is unwise to dump but not using these learnt and accumulated knowledge in a meaningful way. The model learnt in such a way will inevitably lack behind in time. The model is no longer predictive, but chasing the changes at all time.

In [20], Fan presents a simple example to illustrate that old data which is consistent with the new concept can help learning. Instead of throwing away old data, he proposed a framework that dynamically selects one of the following four classifiers as the final classifier:

- 1) The optimal classifier trained so far without the use of latest chunk of data.
- 2) The classifier trained by updating the optimal classifier in 1 with the latest chunk of data.
- 3) The classifier trained from scratch with the latest chunk of data.
- 4) The classifier trained from scratch with the latest chunk of data and some old data samples that are (assumed to be) consistent with latest chunk of data.

The chosen classifier is the one with best accuracy using cross validation on the latest chunk of data. Then, this classifier is set to be the optimal classifier trained so far (Classifier 1) in the next classifier selection process.

The intuition is that no one knows if the latest data chunk on its own is sufficient to train a better classifier than the previous one learnt. Instead of statically defining how much old data to be used, Fan's framework lets the data make the decision dynamically.

In this paper, we begin the dynamic learning framework design with this intuition and show how MOGP can serve as an elegant framework for dynamic learning. Then we propose a novel way to improve the performance of the framework by reducing the error rate and time taken to response to changes.

Since there is not a popular security policy model that supports time-varying policies, we design a time-varying risk budget based security policy model for our experiments. This

model is used to generate training examples for MOGP and serves as the benchmark against which the models learnt by MOGP are evaluated.

III. THE TIME-VARYING, RISK BUDGET BASED SECURITY POLICY MODEL

In a system using a risk budget based security policy, each user is given an amount of risk tokens that represents how much risk the system is willing to take with that user. To access a piece of information, an user offers the amount of risk tokens he is willing to spend from his budget to pay for the access. The system evaluates the risk incurred in granting the access and grant the access only if the user's offer is greater than or equal to the risk. The risk evaluation is part of the policy and may change with time.

The model described in [21] is used for risk evaluation. Risk is defined as the expected value of damage:

$$\text{risk} = (P : \text{probability of incurring damage}) \times (V : \text{value of damage}) \quad (1)$$

The probability of incurring damage, P is decomposed into four *independent* probabilities:

- 1) P_{CH} : The probability that the communication channel between the user and the system is compromised.
- 2) P_{IS} : The probability that the information system is compromised.
- 3) P_{HU} : The probability that the human user is compromised, i.e.; being tempted, malicious, careless, etc.
- 4) P_{PH} : The probability that the physical security of the user or the system is compromised.

It should be noted that P_{CH} , P_{IS} and P_{HU} exclude the probability of physical compromises that are covered by P_{PH} . Whilst the independence assumption among these probabilities might not be true and results in P being slightly overestimated; this is fine from the security perspective, especially given the fact that all these probabilities are only estimates to begin with.

To estimate P_{CH} , we only consider the security level of the communication channel, S_{CH} is either secure ($S_{CH} = 1$) or not ($S_{CH} = 0$). $P_{CH} = 0$ only if $S_{CH} = 1$ and $P_{CH} = 1$ otherwise.

To estimate P_{IS} , we submit to the five level information system security ratings, S_{IS} as outlined in Trusted Computer System Evaluation Criteria (TCSEC). We assume that S_{IS} is an integer in the range $[0, 4]$; the higher S_{IS} , the more secure the system is. S_{IS} is mapped to P_{IS} using an inverse exponential function such that $P_{IS} = 1/\exp(S_{IS})$.

To estimate P_{HU} , we consider the sensitivity labels of the subject (user), sl and the object (information), ol . The sensitivity label of a subject represents the level of trustworthiness of the subject whereas the sensitivity label of an object indicates the level of damage incurred if the object is lost or misused. To map these sensitivity labels to P_{HU} , we reused the sigmoid function as outlined in [16] as follow:

$$P_{HU} = \frac{1}{1 + \exp(-k(TI(sl, ol) - mid))} \quad (2)$$

where $TI(sl, ol)$ 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:

$$TI(sl, ol) = \frac{a^{(ol-sl)}}{M - ol} \quad (3)$$

The intuition for P_{HU} and TI can be found in [16]. The value mid is the value of TI that makes P_{HU} equal 0.5; the value k controls the slope of P_{HU} . The value M is the ultimate object sensitivity level and the temptation TI approaches infinity as ol approaches M ; the intuition is that 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 sl and ol are integers in the range of $[0, 9]$; the settings for k , mid and M are $k = 3$, $mid = 4$, $M = 11$.

To estimate P_{PH} , we assume there are 10 levels of physical security ratings, S_{PH} . S_{PH} takes an integer in the range $[0, 9]$; the higher the rating level, the more secure it is. The mapping function from S_{PH} to P_{PH} used is $P_{PH} = (9 - S_{PH})/9$.

The probability of incurring damage, P is the joint probability of these four probabilities.

To estimate the value of damage, V , the sensitivity level of an object, ol is considered to be the *order of magnitude* of damage, and V can be estimated using an exponential function such that $V = a^{ol}$ [16]. In our experiments a is set to be 10.

In order to introduce dynamic changes to the policy, we further multiply the risk calculated with a safety margin factor, α which varies over time depending on the operational environment. For example, the risk policy can be more restrictive and uses a larger α value at night. We restrict α here to be a real number in the range of $[1.0, 3.0)$. The *evaluated risk* for an access to a piece of information is therefore $\alpha \times P \times V$.

We assume that each user is rationale when making an access request in the following ways. First, each user is able to estimate the risk to a certain degree of accuracy. Second, each user always attempts to minimise the amount of risk tokens spent for each access as well as to maximise his chance in gaining that access.

To model this user behaviour, we assume the user will always make an offer of $(\beta_{min} + \gamma) \times P \times V$, where γ is a random variable with a beta distribution which has a mean value of 0.5 and a variance value of 0.05. The user adjusts its β_{min} overtime based on the allow/deny responses he received. This is achieved by using a counter. The counter increments if an access request is granted and decrements otherwise. After N decisions, the user increases β_{min} by 0.1 if the counter value is positive and decreases β_{min} by 0.1 otherwise. The counter is then reset to zero. The value of β_{min} is initialised to be 0.5 less than the initial value of α .

Examples of access control decisions are generated using the setting described above. Each example is a tuple of the 6 input variables, S_{CH} , S_{IS} , sl , ol , S_{PH} , the *offer* and the *decision*. We generate a set of 10000 examples. For each set of examples, the value of α is changed randomly within its range every 1000 examples. Using these examples as a training set, MOGP is used to learn an access control policy from them.

IV. MULTI OBJECTIVE GENETIC PROGRAMMING

A. Evolutionary Algorithms

Evolutionary Algorithms (EA) are a family of problem solving techniques inspired by natural selection¹. An initial population of individuals is generated, typically in a random fashion. Each individual represents a candidate solution to the problem in question.

The population of individuals is evolved repeatedly in the following way to produce new generations of population that offer increasingly better solutions to the problem.

First, each individual in the population is evaluated and given a fitness score that measures how well the individual solves the problem. Then, the population of individuals is subject to the evolutionary operators to produce the population of the next generation as follows:

- Selection: individuals are selected for breeding according to fitness (natural selection). This is an implementation of the “survival of the fittest” concept, ensuring better (fitter) individuals are more likely to contribute the next generation;
- Crossover: parts of two individuals are exchanged to form two new individuals;
- Mutation: elements within an individual is perturbed in some way. This serves to diversify 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.
- Reproduction: an individual is passed on to the next generation unchanged.

Commonly, the stopping criterion is either that a “good enough” solution (individual) has been found or that a preset number of generations have been produced.

B. Genetic Programming

Genetic Programming (GP) is a form of EA wherein an individual is a program represented by a tree structure. An example that implements the formula $(X \times Y) + (4 - Y)$ is shown in Figure 1.

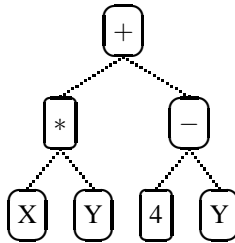


Fig. 1. An example individual in GP

The nodes in such a 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

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

(the non-leaf nodes). An example terminal set T and function set F sufficient to allow description of the LISP tree in Figure 1 is given below as:

$$T = \{X, Y\} \cup \{1, 2 \dots 100\}$$

$$F = \{+, -, \times, \div, \%\}$$

In choosing these sets, there are two properties that must be ensured: the *sufficiency* property and *closure* property. 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 they 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 [22].

The *crossover* and *mutation* operators in GP are defined as follows [3]:

- Crossover is performed on two trees. A sub-tree in each tree is chosen randomly and swapped with the other.
- 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.

C. Multi Objective Evolutionary Algorithm (MOEA)

In many practical problems, a desire to optimise more than one objective is common. Some of these can conflict with one another. In other words, instead of having a single fitness score, each possible solution has a vector of fitness scores, one per objective. This is typically referred to as a multi objective optimisation problem. The traditional way to approach this is to aggregate the vector of fitness scores using a weighted sum approach. In a sense, this is like “comparing apples with oranges” by weighting them differently. There is inevitably some degree of subjectiveness and arbitrariness in weight assignments. A more principled and conceptually clear approach would be advantageous.

MOEA uses the well known concept of Pareto dominance in the fitness evaluation of each solution (individual) in the population. A Pareto Dominance, \succ relation between two solution x and y is defined as follow:

Pareto Dominance, \succ . Let f_x and f_y be the fitness vector of x and y , $x \succ y \Leftrightarrow \forall i \cdot f_x[i] \geq f_y[i] \wedge \exists i \cdot f_x[i] > f_y[i]$ where $f[i]$ is the fitness score of the i -th objective.

In other words, x dominates y iff x is better than y in at least one aspect and is at least equally good in all other aspects. This defines a partial order relation on the solution space. This relation implies there exists a subset of solutions that are not dominated by any other solutions. This subset represents the best solutions possible and is known as the *Pareto Front* or *Pareto Optimal set*.

The aim of MOEA is to converge the individuals in the population to the Pareto Optimal set of solutions. Within the

population of each generation, there is always a subset in which solutions are not dominated by any other solution in the population. MOEA aims to make this *non-dominated* subset a better approximation of the Pareto Optimal set than one in the previous generation. To have a good approximation to the Pareto Optimal set, MOEA also attempts to maximise the diversity among the solutions in the “non-dominated” subset. Consider an optimisation problem with 2 objectives as in Figure 2, let the curve line represent all the solutions with the best possible achievable trade-off between the 2 objectives, i.e. the real Pareto Front, MOEA attempts to converge the individuals (points) in the population to be as near to the Pareto Front as possible and also as diversely spreaded on the Pareto Front as possible.

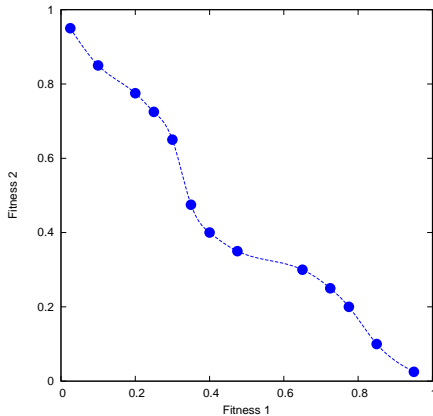


Fig. 2. The ultimate goal of MOEA is to obtain the best approximation of the Pareto Front of the problem in question with the individuals in the population.

In this paper we used Strength Pareto Evolutionary Algorithm (SPEA2), one of the most popular MOEAs to perform multi objective evolution in MOGP. For details on SPEA2, refer to [23].

D. Advantages of MOEA

The Pareto Optimal Set approach used by MOEA has several advantages over the traditional weighted sum approach:

- The weights that define the trade-off among different objectives is no longer required to be determined a priori. Such a determination is hard as it requires a deep understanding of the problem domain.
- The Pareto front can unfold the relationship between different objectives, which may be difficult to obtain otherwise. Such information is also helpful in guiding a decision-maker to choose the optimal solution for the problem from the Pareto Optimal set.
- The set of Pareto Optimal solutions can be saved and be retrieved later. Such a retrieval may be necessary if a change in circumstance requires a different trade-off and therefore a different solution.
- Optimising for multiple fitness scores tends to preserve the diversity of the population, which prevents the popula-

tion from being trapped in a local optimum and increases the chance of finding better solutions.

V. EXPERIMENTAL SETUP

We carried out multiple experiments of using GP/MOGP to learn security policies from decision examples generated according to the time-varying risk budget based security policy model as described in Section III. This model can be viewed as a function that maps a 6 decision factors, S_{CH} , S_{IS} , sl , ol , S_{PH} , $offer$ to a binary decision. To use GP to search for a policy, we choose to use each individual tree in the population to represent a policy. It follows that the words “individual” and “policy” are used interchangeably in the rest of this paper. The terminal (leaf) nodes can be one of the decision factors or an Ephemeral Random Constant (ERC)², which takes a real number in $[-10, 10]$. The non-terminal (non-leaf) nodes are mathematical functions. The functions chosen are: $+$, $-$, \times , \div ³, $protectedln(x)$ ⁴, $\exp(x)$, $pow(x, y)$, $protectedlog_x(y)$ ⁵, $\max(x, y)$, $\min(x, y)$, $\sin(x)$ and $\cos(x)$.

In each experiment, a policy is learnt from a training set of decision examples. Ideally, the learnt policy should output the same decisions prescribed by the examples in the training set when the policy is fed the tuple of 6 decision factors $\langle S_{CH}, S_{IS}, sl, ol, S_{PH}, offer \rangle$ in the examples. The practical objective is to minimise the percentage of output decisions that are different from the true model that generates the examples. This percentage is the *error rate* of the learnt policy. This error rate is estimated using the 1000 examples. We will show that using MOGP actually helps in better achieving this single objective.

All the experiments are carried out using ECJ 18 [24] with the SPEA2 module is obtained from the ECJ 19 CVS repository <http://dev.java.net/>⁶. Unless otherwise specified, the default parameters⁷ were used in all experiments.

As GP (and EA in general) is stochastic in nature, the evolution process in each run varies a lot depending on the random seed used. To see how well GP can perform in learning policies, we carried out 100 runs for each experiment using a different random seed for each run, and evaluated GP’s performance using the best individuals produced by each run. The best individual produced by a run is the individual with the lowest error rate in the last generation of that run. An GP’s performance is evaluated by two measurements:

²Ephemeral Random Constant (ERC) is a constant in which its value is randomly generated during its creation.

$$^3x \div y = \begin{cases} \frac{x}{y} & \text{if } y \neq 0 \\ 1 & \text{otherwise} \end{cases}$$

$$^4protectedln(x) = \begin{cases} ln(|x|) & \text{if } x \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$^5protectedlog_x(y) = \begin{cases} log_{(|x|)}(|y|) & \text{if } x \neq 0 \text{ or } 1 \text{ or } y \neq 0 \text{ or } 1 \\ 0 & \text{otherwise} \end{cases}$$

⁶The reason is that there is a major revision on the module in version 19 to improve clarity and remove some minor bugs found

⁷`koza.params` for Single Objective GP and `spea2.params` for MOGP

- 1) The median error rate of the best individuals. Median is used here instead of mean as the error rate distribution is highly skewed. This follows that the confidence interval based upon standard deviation of mean is no longer valid. Instead, the 95% confidence interval of the median is calculated using the Thompson-Savur formula presented in [25].
- 2) The number of the best individuals with error rates $\leq 25\%$.

These measurements give an indication how low the error rate could be and on the likelihood of GP producing a low error rate policy.

VI. LEARNING STATIC SECURITY POLICY

To prepare for the experiments on dynamic policy learning, we started with three experiments to learn static policy. The training used is the first 1000 examples generated as described in Section III. The policy to be learnt is static in the sense that all 1000 are generated with the same value of α . Experiment 1 uses Single Objective Genetic Programming. Experiment 2 and Experiment 3 use MOGP-SPEA2 to address the problems encountered in Experiment 1.

In the first experiment, the default genetic operators and parameters are used: each selected individual (policy) has a probability of 0.9 to be applied the crossover operator and a probability of 0.1 to be reproduced. To breed the individuals of the next generations, the binary tournament selection scheme [26] is used. The fitness of a policy is its error rate. Let r_i be the 6-tuple in example i , $decision_i$ be the decision in example i , and $P(r_i)$ be the decision of a policy P on r_i , then the fitness function of P is:

$$f_{all}(P) = \frac{1}{n} \sum_{i=1}^n P(r_i) \neq decision_i \quad (4)$$

In this case, $n = 1000$ and the numbers 1 and 0 are used to represent Boolean values *True* and *False*.

The experiment is carried out with all 100 runs terminated after 200 generations. The median error rate of the 100 best policies produced by these runs is 0.3555 with its 95% confidence interval of [0.3487, 0.3640]. The best of the best individual has an error of only 0.107 and there is only 12 out of the 100 best policies have error rate ≤ 0.25 . Worse, more than half of these best policies have error rates > 0.35 . The distribution of the error rates of these best individuals is shown as Experiment 1 in Figure 3. This suggests that many of the runs got stuck at local optima. Analysis on the tree structure of policies in the population reveals some common problems in Genetic Programming. We now present these problems and the methods used to alleviate them.

A. Increase in average individual size and evaluation time

The size of an individual is the number of nodes in its tree representation. The average size of the individuals in the population grows quickly and becomes very large. This can be either a phenomenon of bloat (uncontrolled growth of the average size of the individuals) or over-fitting problems, or

even both. We do not attempt to distinguish them here as they both make the learning process more difficult. Furthermore, as the size of the individuals become larger, they consume more memory and require longer time to evaluate.

To alleviate these problems, we use SPEA2 bloat control which has been shown to be effective [27]. This method introduces a new objective to the experiment — minimising the individual size. Let $size(P)$ be the size of an individual P , the fitness function with respect to individual size is:

$$f_{size}(P) = \begin{cases} size(P)/512 & \text{if } 32 \leq size(P) \leq 512 \\ 32/512 & \text{if } size(P) < 32 \\ 1 & \text{otherwise} \end{cases} \quad (5)$$

In other words, individuals with less than 32 nodes have the same f_{size} values as the individual with 32 nodes and individuals with more than 512 nodes have the same f_{size} values as the individual with 512 nodes. This is to avoid over-simplified or over-complicated solutions.

As SPEA2 maintains an archive that ensures the individuals in the non-dominated subset of a generation are copied to the population of the following generation, the reproduction operators is removed, i.e. the crossover operator is applied with probability of 1.0.

The second experiment is carried out with these changes. The results show the countermeasure is effective. The average size of the individuals is significantly smaller and the evaluation time is also much faster. However, the performance improvement in terms of error rate is very marginal. After 200 generations, the median error rate of the 100 best policies is 0.3545 with its 95% confidence interval of [0.3419, 0.3591]. The best of the best individual has an error of only 0.126. The number of best policies that have error rate ≤ 25 increases to 22. The error rate distribution of these best individuals is shown as Experiment 2 in Figure 3.

B. Loss of diversity among constants and individuals

The diversity among constants appearing in the individuals decreases with each new generation. This is expected as some forms of convergence is necessary if a population is to produce a solution for the problem in question. It is often that the desired constant will not appear in the initial population that is randomly generated; but the evolutionary process is expected to synthesise the required constant by joining the existing ones through the operators. However, if the constants converge prematurely before finding the appropriate one, the evolution process will be stuck at a local optimum.

This lack of diversity among constants is the key to our problem. This is revealed by analysis on the risk-based policy discussed in Section III. This model can be written as:

$$\begin{aligned} allow \text{ iff } offer &\geq risk \\ \Rightarrow allow \text{ iff } offer &\geq \alpha(P \times V) \\ \Rightarrow allow \text{ iff } offer &\geq \alpha \times P \times a^{ol} \end{aligned} \quad (6)$$

As P is in the range $[0, 1]$ and α is in the range $[0, 3]$ and $offer$ is programmed to track $risk$, the value of constant a

which is raised exponentially dominates the right hand side of inequality (6). An individual would still have a high error rate even if it otherwise exactly implements the inequality but with the wrong value for a . As the diversity of the constant decreases over generations, the chance of finding the correct value of a becomes even smaller.

In our experiments, the rate of convergence among constants is accelerated by the following factors:

- Small chance of using a constant as a leaf node — In ECJ, each ERC constant of a range is implemented as a terminal class and each variable is implemented as a terminal class. Each terminal class has an equal probability to be selected to be a leaf node. As our experiments consists of 6 variable terminal classes and 1 ERC terminal class, the probability of an ERC to be chosen as a leaf node is only $1/7$. Furthermore, the relatively large ERC range also exacerbated the problem in finding the appropriate constant.
- No mutation operator — Without mutation, there is no new constant introduced to the population at all. The diversity among constants in the population is decreasing in every generation.
- The use of archive in SPEA2 — SPEA2 restricts the binary tournament selection among the individuals in the archive and therefore only the constants that appear in these individuals have the opportunity to be passed to the next generation.

Further analysis also reveals that substantial portion of the individuals in the archive share the same or similar higher level structures of their trees (assuming root of the tree is the highest node). The diversity among the individual is lost. We think the cause of this problem is similar; the absence of mutation and the use of archive in SPEA2.

To overcome these problems, we setup the third experiment with the following changes. First, we used six identical ERC classes instead of just one. This makes the probability in choosing between a variable and a constant become equal. Second, we choose to use mutation operator only instead of the typical high crossover and low mutation settings. There are four reasons to do this. First, mutation introduces new gene and thus promotes diversity in the population. Second, it has been shown empirically that crossover does not outperform mutation in many problems in GP. Third, this frees us from tuning the probability parameter of applying crossover and mutation. Finally, mutation provides a means to introduce new individual to the population by simply allowing mutation to happen at the root node of an individual tree. We changed the probability of mutation at the root node from 0 (default) to 0.125, at a terminal node from 0.1 (default) to 0.125 and at a non-terminal nodes from 0.9 (default) to 0.75. The value 0.125 is chosen based upon the ratio between the archive size (128) and the population size (1024) used in ECJ.

The results show a significant performance improvement in terms of error rate and success rate. After 200 generations, the median error rate of the 100 best individuals reduced to 0.2225 with its 95% confidence interval of $[0.1595, 0.2789]$. The best

of the best individual has an error of only 0.099. Also, the number of best policies with error rates $\leq 25\%$ became 54. The error rate distribution of these best individuals rates is shown as Experiment 3 in Figure 3.

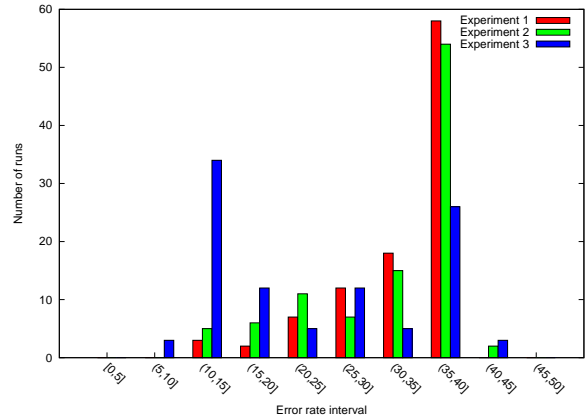


Fig. 3. The distribution of the best individuals in the 100 runs from each of the first three experiments on each error rate interval.

VII. LEARNING DYNAMIC SECURITY POLICY

In the first three experiments presented so far, all the decision examples are available prior to the start of the learning process. This is hardly the case in practice. In this section, we investigate how to learn dynamically when the decision examples become available gradually during the course of learning. The learnt model is continuously refined or even redefined if necessary by the new decision examples. This section presents three experiments: Experiment 4 shows how dynamic learning can be done by extending the previous static learning algorithm. Experiments 5 and 6 use a novel dynamic learning framework — Diversity via opposite objective (DOO).

In all experiments, the input decision examples are organized into a sequence of data chunks. Each chunk consists of 200 examples. The risk-based policy model described in Section III is used to generate the data. The policy is changed every 1000 examples by changing the value of α . In each experiment, the sequence is fed into a MOGP learning process one chunk at a time. The first policy is learnt using the first chunk after 100 generations of evolution. Each subsequent chunk is used by the learning process to refine the policy learnt from the previous chunk(s). Each refinement starts with the population of the last generation learnt from the previous chunk(s), and uses the examples in the latest chunk to learn a refined policy after 100 generation of evolutions.

A. Experiment 4: Dynamic Learning framework based on Fan’s intuition

In Experiment 4, the previous learning framework is extended in two ways. First, the number of examples, n in $f_{all}(P)$ is no longer fixed to 1000 but set to be the total number of examples received; n increases as more chunks

are received. Second, a new fitness function is introduced to measure the error rate of a policy P with respect to the latest chunk of data. Let r_i be the 6-tuple in an example i , $decision_i$ be the decision in example i , s be the size of a data chunk, and $P(r_i)$ be the decision made by P on r_i , this new fitness function, f_{last} is:

$$f_{last}(P) = \frac{1}{s} \sum_{i=n-s+1}^n P(r_i) \neq decision_i \quad (7)$$

With the introduction of this new fitness function, this experiment has 3 fitness functions in total, namely f_{all} , f_{last} and f_{size} .

The intuition employed here is similar to Fan’s (refer Section II-B). Each chunk is used to refine the policy learnt from the previous chunks. The refinement process happens through 100 generations of MOGP learning. In each generation:

- 1) The policies with the lowest f_{all} or f_{last} values in the previous generations are preserved in the SPEA2 archive. These policies correspond to Fan’s Classifier 1.
- 2) These policies are refined with examples in the latest chunk through MOGP. These refined policies correspond to Fan’s Classifier 2.
- 3) New policies are generated (using mutation). Some of these policies will have the lowest f_{all} value within the population and correspond to Fan’s Classifier 3. Some others will have the lowest f_{last} value and correspond to Fan’s Classifier 4.

After 100 generations, the policy with the lowest f_{last} value is chosen to be the new learnt policy.

The experiment is carried out with these settings. The results are shown in two levels of resolution. Figure 4(a) shows the median error rate of the best policies learnt after every 100 generations of training time. Except in the initial 500 generations, the median error rate is kept below 0.220 at all time.

To understand the learning progress in detail, Figure 4(b) shows the median error rate of the best policies in each generation. When policy change happens at every 5 chunk of decision examples, i.e. $5 \times 100 = 500$ generations, the median error rate spikes up sharply. After the spike, the error rate reduced faster than the initial 500 generations. This is a direct effect of using the 2 fitness functions: f_{all} and f_{last} together. It provides a smooth transition phase for learning by protecting the models that are optimal with respect to the old policy from being eliminated quickly by the new policy. Consequently, these models are able to pass on the knowledge they learnt to the new individuals created after the change. Consider the population in the generation prior to the change, the models that have low f_{last} values are also likely to have low f_{all} values. After the change, their f_{last} will become worse (higher), but their f_{all} values would only be affected slightly. Thus, these models will still have a good chance to be kept in the archive and be brought forward to the next generation. As the policy change in our problem is characterised by a change on the value of α , the learning of a

new policy is thus reduced to learning of a new value for α . This is obviously simpler than learning from scratch.

Further analysis also reveals the individuals in the archive converge and become more alike to one another over generations. For our problem, this is not an issue as long as a policy is learnt before the diversity is lost. The subsequent changes only require changes on the value of α which the mutation operator can easily provide. The loss of diversity also explains the very sharp upward spikes during policy change in the error rate in Figure 4(b). Since the individuals are all alike, none of them would be a good match for a new policy when a policy change happens and thus the error rate increases sharply. However, the new policy is relatively easy to learn and thus the quick decrease in error rate.

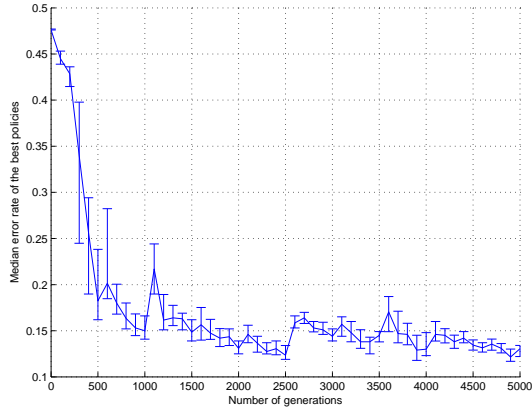
In summary, this dynamic learning approach can perform well under the assumption that the policy change is relatively small and the knowledge learnt previously can aid the learning of the new policy. The use of 2 fitness functions: f_{all} and f_{last} protects and allows the knowledge learnt to be passed on to the next generation. However, this approach still suffer from the loss of diversity among the individuals. This puts the general applicability of this approach in question.

B. Diversity via Opposing Objectives (DOO) Framework

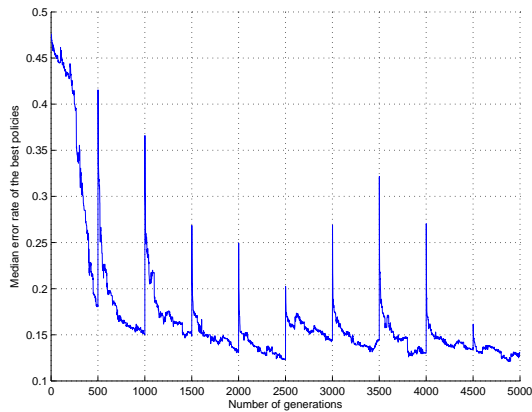
The loss of diversity problem is not uncommon in the traditional EA settings. The “survival of the fittest” principle employed by EA provides the fitter individuals higher chances to survive and pass their genes to the individuals in the next generations. Consequently, the individuals in the population will inevitably become more alike with one another over generations, i.e. the diversity among individuals will be lost over generations. If the diversity is lost prior to the optimum individual is found, the individuals in the population is said to be trapped in a local optima. To prevent being trapped at local optima, EA uses mutation operators to introduce new random genes to the individuals. However, as these genes are generated randomly, the chances that they provide improvement over current individuals are very small and therefore are highly unlikely to be preserved. In other words, the diversity is generated and then lost from one generation to the next.

To overcome this problem, several EA-based dynamic learning algorithms have been proposed in the literature. Most, if not all of them *first* attempt to produce individuals that are optimised for the problem related objectives and *then* attempt to maintain the diversity among individuals in the population as much as possible [28]. Their settings are often ad-hoc and the algorithms are often complicated. We propose a new dynamic learning framework — Diversity via Opposing Objectives (DOO). DOO takes the opposite perspective; it *first* attempts to maximise the diversity among individuals in the population through generations, and *then* uses the ever increasing diversity to help in finding optimised individuals.

Performing evolutionary operations on a single individual in EA can be viewed as searching for more optimised individuals from the position of the individual in the solution space. Performing evolutionary operations on a diverse population of



(a) The median error rate of the best policies after 100 generations training time for each chunk of decision examples



(b) The median error rate of the best policies

Fig. 4. The experimental results of dynamic learning using 3 objectives

individuals can be viewed as searching many different parts of the solution space in parallel. This parallel search has a much better chance of finding more optimised individuals than a search starting from just one individual. The ever increasing diversity in DOO results in a domino effect such that not only the search is done in parallel, but the search space coverage increases as the diversity increases. Consequently, the chance of finding a more optimal individual becomes larger and larger as the evolution proceeds. This same effect cannot be achieved by conducting many runs of single-objective EA in parallel because each run is likely to be trapped in a local optima.

DOO is very simple. DOO changes each and every objective into a pair of opposing objectives. For example, the objective of minimising error rate is changed to minimising error rate and minimising accuracy ($1.0 - \text{error rate}$). DOO then optimises all objectives using MOEA. The opposing objectives in DOO ensures that an individual who is less fitter for one objective is fitter the opposing objective. Therefore, *no individual is dominated by others*, i.e. all individuals are at Pareto Front. Therefore, each and every individual has

a fair chance of passing its genes to the next generation, and evolutionary operations will produce more diversity in the next generation. Using the (error rate, accuracy) pair of objectives as an example, a possible Pareto Front formed by the individuals is depicted in Figure 6. From an MOEA's point of view, since the true Pareto Front is already found at the start of the evolution process, the only job left for MOEA is to improve the spread of the solutions on the Pareto Front (See Section IV-C for details). Therefore, the population of a generation has a wider coverage of the solution space than the previous generations, i.e. diversity increases as MOEA drives the evolution process.

To understand how DOO works via MOEA, we introduce the concepts of the solution space S and objective space O . Referring to Figure 5, each point $s \in S$ represents a possible solution to the problem in question. The fitness function, f maps a point $s \in S$ to a point $o \in O$ such that the location of the point o represents how well s meets the objectives. The function f is surjective (onto) but not injective (1 to 1); each point in S is mapped to a point in O and multiple points in S can be mapped to one point in O .

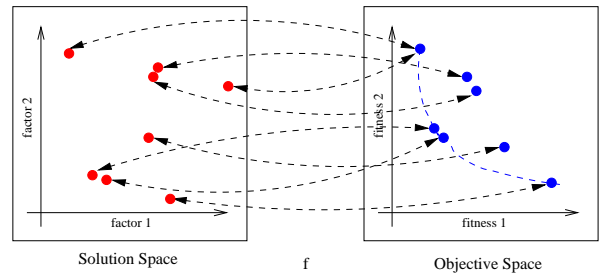


Fig. 5. Mapping between solution space, S to objective space, O .

Diversity among individuals in a population is essentially a measure on how uniformly the solution points are distributed in S . While f does not conserve the distribution, it is often true that f is a continuous function between these two topological spaces, i.e. a set of solution points near a point s' in S is mapped to a set of point near the point $f(s')$ in O .

As f is not injective, the inverse is not necessarily true, e.g. 2 solutions can be very different yet both solve the same problem equally good. However, following the continuity assumption on f , it is reasonable to assume that a set of points in O that are far apart from one another corresponds to a set of points that are far apart from one another in S . DOO makes use of this assumption and attempts to maximise the diversity of solution points in S via maximising the diversity of points in O using MOEA.

In the SPEA2 implementation of MOEA, if the number of non-dominated points exceeds the archive size, the point that has the shortest Euclidean distance to its k -th nearest neighbor is dropped, where k is usually set to be the square root of the sum of the the population size and the archive size [23]. If two points have the same distance to its k -th nearest neighbor, then the tie is broken by comparing

their distances to their $(k - 1)$ -th nearest neighbors and so forth. This process is iterated until the non-dominated points can fit into the fixed size archive. Essentially, the goal is to fill the archive with non-dominated points as uniformly and widely distributed over O as possible. Using the \langle error rate, accuracy \rangle pair of objectives as an example, a possible archive is depicted in Figure 6. As the most optimum point of each objective is always located at the corner point of the convex hull of the Pareto Front formed, i.e. they are furthest apart from others. These points are guaranteed to be preserved at each generation until a better one is found; and a better solution is likely to be found as the diversity keeps increasing.

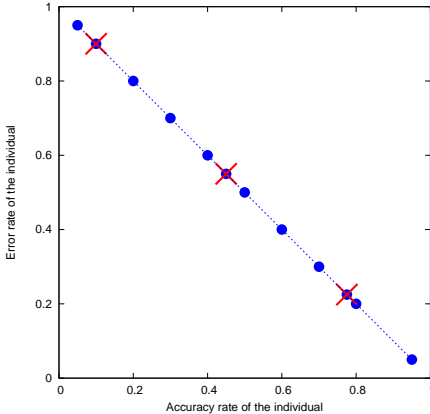


Fig. 6. The use of opposing objectives implies that every individual in the population is a Pareto Optimal solution. SPEA2 truncates individuals that are close together *iteratively* until they can fit into the archive. Assuming the archive size is 10, the individuals removed is illustrated. It should be clear to see that the individuals at both corner of the Pareto front are guaranteed to survive from this truncation process

Furthermore, every policy that output binary decisions (all binary classifiers in general) can be inverted to its complement by a simple negation on the output decision. Therefore, high error rate policies are just as good as those with the low error rate. To gain benefit from this, final output policy selected by DOO is the one with highest absolute value of bias; bias is defined as $0.5 - \text{error rate}$. A negative bias value implies that the policy is optimised on the opposing objective and thus its output decisions need to be flipped if it is selected for use.

It may be argued that another way to preserve diversity is to randomly select the individuals whose genes will be passed to the next generation. Our experience shows that this may next work in practice. Our experimental results indicate that the error rates of the randomly generated individuals of the first generation cluster around 50%. Such a distribution may be attributed to the law of large number. Randomly selecting individuals to pass on their genes means that the error rates of the next generation are likely to cluster around 50% as well. DOO creates two opposing forces to pull this cluster apart.

C. Experiment 5: Two Pairs of Opposing Objectives

To use DOO, Experiment 4 is modified to use two pairs of opposing objectives: $\langle f_{all}, 1.0 - f_{all} \rangle$ and $\langle f_{last}, 1.0 - f_{last} \rangle$.

The fitness of the individual size, f_{size} is excluded here as it is not a problem related objective.

The experiment is carried out with all other settings remain the same. The results are presented in two levels resolutions as before. Figure 7(a) shows the median error rate of the best policies learnt after 100 generations of training time. Figure 7(b) shows the median error rate of the best policies in each generation. The results of Experiment 4 using 3 objectives is also included in pink for comparison purpose.

In the initial 500 generations, the learning rate in DOO is significant faster. The median error rate of the best policies is reduced to 0.250 in 200 generations. This result is comparable to the best static learning approach in Experiment 3, despite the fact that the amount of training examples used is significantly less (400 vs. 1000) and presented to the learning algorithm in sequential chunks.

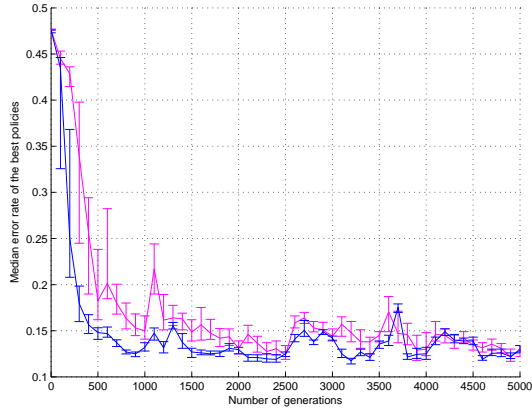
Furthermore, the best policies learnt in each generation using DOO generally has a lower or equal median error rate than the one learnt in previous experiments. However, the error rate can suddenly rise even in the absence of a policy change. Analysis reveals that this is related to the model selection problem. Currently, the output policy is selected based upon the *estimated* error rate/bias on the latest 200 examples. As these 200 examples are generated randomly, they may not be sufficient in forming a good representation of the target policy. Moreover, changes in policy sometimes can simply mean a revisit of an old policy in the past. Therefore, the policy that is optimal in respect to the latest chunk may not be the true optimal policy. This sub-optimal policy selection effect is not obvious in previous experiment as the policies are very much alike to one another. We will show how the ensemble approach can be used to overcome this problem in the Section VIII. From the other perspective, this effect is a positive sign of diversity maintenance.

Lastly, the heights of the spikes in the error rate due to policy changes are much lower in DOO. This is another evidence of diversity maintenance. One could view policy change as a change on the fitness mapping function, f . With a diverse set of policies maintained in the population, it is likely that one of them will be near the new target policy after the change.

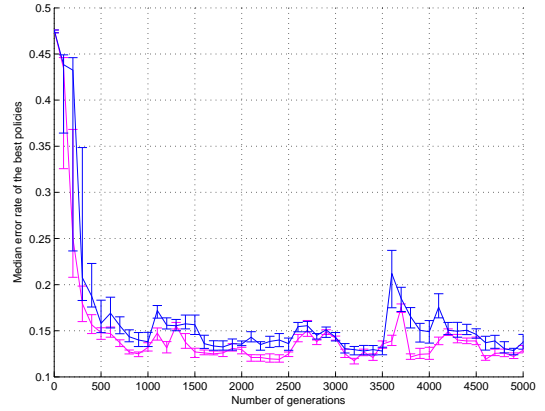
D. Experiment 6: One Pair of Opposite Objectives

An obvious weakness of the previous experiment is that it is not scalable. The evaluation of f_{all} involves scanning through all the decision examples seen. As the number of examples increases over time, the fitness evaluation time required for each individuals follows. A possible way to overcome this is to use a subset of decision examples, randomly sampled from all the decision examples seen. The likelihood of an example being sampled decay over time, i.e. the older an example is, the less likely it is being sampled.

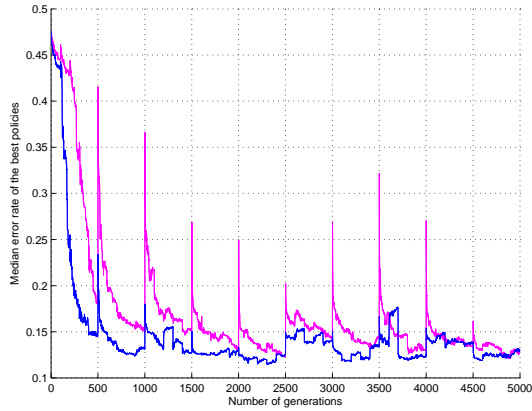
In Experiment 6, we drop the pair of objectives $\langle f_{all}, 1.0 - f_{all} \rangle$ from Experiment 5 to see the effect of not evaluating fitness against older examples.



(a) The median error rate of the best policies after 100 generations training time for each chunk received

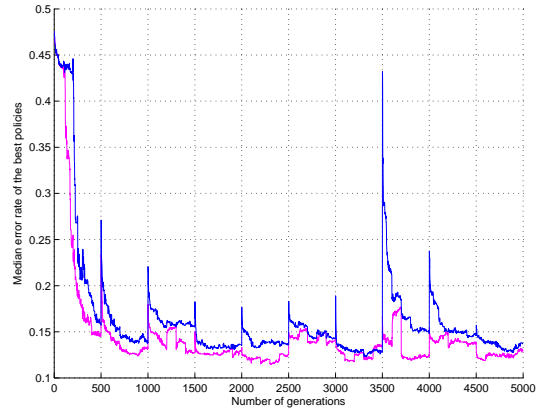


(a) The median error rate of the best policies after 100 generations training time for each chunk received



(b) The median error rate of the best policies

Fig. 7. The results of experiment with 2 pairs of opposite objectives



(b) The median error rate of the best policies

Fig. 8. The results of experiment with 1 pair of opposite objectives

The experiment is carried out with only this change. The results are shown in two figures: Figure 8(a) shows the median error rate of the best policies learnt after every 100 generations of training time and Figure 8(b) shows the median error rate of the best policies in each generation. The results of Experiment 5 that uses 2 pairs of opposing objectives is also included in pink for comparison purpose.

The experimental result shows that the performance of the best policies obtained in this experiment lay somewhere between those obtained in Experiment 4 and those obtained in Experiment 5. This suggests that the the pair of objectives $\langle f_{all}, 1.0 - f_{all} \rangle$ dropped are actually useful in maintaining a better diverse set of individuals in the populations. It remains to be our future work to investigate the best way to investigate this further as well as seeking a sampling technique to select the old data to make the framework scalable whilst maintaining the performance.

VIII. MODEL SELECTION

In this section, we examine various ensemble approaches in which the output model can be constructed by combining mul-

iple models to achieve better performance. We use a simple voting mechanism such that the output of the ensemble is the majority output all the models. This ensemble construction is virtually no cost in EA by simply selecting the best N individuals from the final population.

However, the theoretical study of ensembles has revealed the 2 key factors that determine the performance of an ensemble are the performance of individual models and the diversity among all models in the ensemble [29]. As the population of an EA run with non-DOO setting converges and loses the diversity among individuals, the performance gain of using ensemble is limited to the first factor. However, this is not a problem with DOO. Yet, we still have 2 questions to answer:

- 1) How many models should be used to construct the ensemble? Whilst the negative bias models are as useful as the positive bias models, models with near zero bias are virtually useless. Should these models be included? If not, what should be the threshold on the bias of a model such that it is included in an ensemble?
- 2) Should the vote among these models be weighted?

We choose to examine if weighting the vote of each model with its bias on the latest chunk is better than a simple uniform weighted vote. If so, how much is the improvement?

We attempt to answer these questions by comparing the performance of ensemble built with the following combinations of models:

- use the single highest bias model in the archive.
- use the 8 highest bias models in the archive.
- use the 16 highest bias models in the archive.
- use the 32 highest bias models in the archive.
- use the 64 highest bias models in the archive.
- use all models (128) in the archive.

in which the bias is estimated with the bias of the model on the latest chunk of examples.

The models in each of these ensembles are combined with 2 different methods: uniform weighted (unweighted) and bias weighted voting mechanisms. In bias weighted voting mechanism, the vote of each model is weighted with the absolute value of its bias on the latest data chunk. If the bias is negative, its vote goes to the complement decision class.

When using the ensemble approach in Experiment 4, the error rate does not decrease but increases with the number of models used as shown in Figures 9(a) – 9(b). This is because not all the models are optimised on error rate, some models in the archive are optimised on other objectives, e.g. the model size. When the number of models used is small (8 or 16 models), it is still very likely that the selected models are those optimised on error rate. However, as these models have converged to become very similar to one another, the use of ensemble does not result in any performance gain nor loss. As the number of models used increases, those models that are not optimised on error rate are also included in the ensemble. The performance of the ensemble become worse. The deterioration in performance is worse in the unweighted voting mechanism.

The results on using the ensemble approach in Experiment 5 and 6 are shown in Figures 9(c) – 9(f). When the number of models used is small (8 or 16 models), there is no significant change in performance. However, the error rate curve becomes smoother, the sudden rises happen in the absence of policy changes seem to disappear. This smoothing effect is especially clear between generations 2500 to 3000 and also between generations 3500 to 3700. As the number of models used increases, the performance only becomes slightly worse. The diversity maintained in DOO provides performance gain to counter the performance loss due to the use of less bias models in the ensemble.

IX. CONCLUSION AND FUTURE WORK

This papers shows that dynamic policy can be learnt from examples using evolutionary algorithms. We propose a novel dynamic learning framework — Diversity via Opposite Objectives (DOO). DOO treats an N objectives optimisation problem as $2N$ objectives optimisation problem by adding

an opposing objective for each of the original objectives. With such a setting, DOO is able to maintain the diversity among the individuals in the population whilst optimising the intended objectives. Diversity among individuals can aid in avoiding premature convergence and coping with concept drift in dynamic learning.

In our future work, it is envisaged to investigate DOO framework in more detail. This includes examining the use of random sampling on old decision examples to overcome its current weakness, extension on replacing its current GP representation with other techniques such as neural network and decision tree.

ACKNOWLEDGEMENT

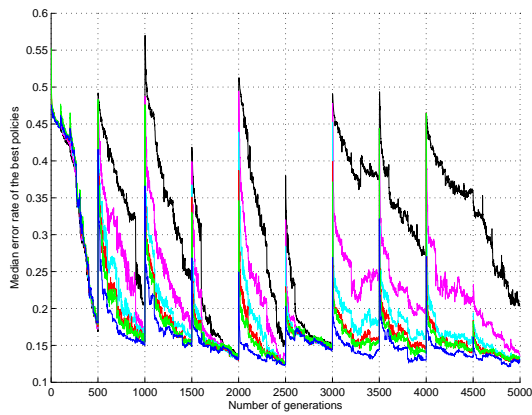
Research was sponsored by US Army Research laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either 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.

We would also like to thank Wei Fan, Charanjit Jutla and Charu Aggrawal from IBM Watsons Research Center for providing useful direction and feedback on the work.

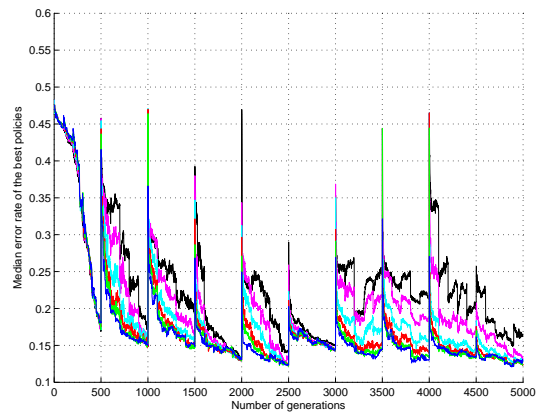
REFERENCES

- [1] “Horizontal Integration: Broader Access Models for Realizing Information Dominance,” The MITRE Corporation JASON Program Office, Mclean, Virginia, Tech. Rep. JSR-04-132, Dec 2004.
- [2] Y. T. Lim, “Survey on Risk-based Decision Making,” University of York, York, United Kingdom, Qualifying Dissertation, July 2007.
- [3] J. R. Koza, “Hierarchical Genetic Algorithms Operating on Populations of Computer Programs,” in *IJCAI*, 1989, pp. 768–774.
- [4] Y. T. Lim, P. C. Cheng, J. A. Clark, and P. Rohatgi, “Policy Evolution with Genetic Programming: a Comparison of Three Approaches,” in *2008 IEEE Congress on Evolutionary Computation*, IEEE Computational Intelligence Society. Hong Kong: IEEE Press, 1-6 Jun. 2008, pp. 813–819.
- [5] —, “Policy Evolution with Genetic Programming,” IBM Research Report RC24442, Tech. Rep., 2008.
- [6] —, “MLS Security Policy Evolution with Genetic Programming,” in *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*. Atlanta, Georgia, USA: ACM Press, 12-16 Jul. 2008.
- [7] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, “Mining data streams: a review,” *SIGMOD Rec.*, vol. 34, no. 2, pp. 18–26, 2005.
- [8] C. C. Aggarwal, *Data Streams: Models and Algorithms (Advances in Database Systems)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.
- [9] D. Barbara, *Applications of Data Mining in Computer Security*, S. Jajodia, Ed. Norwell, MA, USA: Kluwer Academic Publishers, 2002.
- [10] P. D. McDaniel, “Policy Evolution: Autonomic Environmental Security,” December 2004.
- [11] M. Kuhlmann, D. Shohat, and G. Schimpf, “Role mining - revealing business roles for security administration using data mining technology,” in *SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies*. New York, NY, USA: ACM, 2003, pp. 179–186.
- [12] N. Li, T. Li, I. Molloy, Q. Wang, E. Bertino, S. Calo, and J. Lobo, “Role mining for engineering and optimizing role based access control systems,” Tech. Rep., 11 2007.

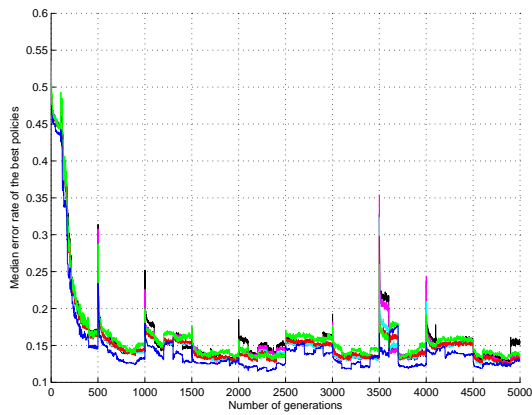
- [13] J. Schlegelmilch and U. Steffens, "Role mining with orca," in *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies*. New York, NY, USA: ACM, 2005, pp. 168–176.
- [14] A. Tongaonkar, N. Inamdar, and R. Sekar, "Inferring higher level policies from firewall rules," in *LISA'07: Proceedings of the 21st conference on Large Installation System Administration Conference*. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–10.
- [15] Y. T. Lim, P.-C. Cheng, J. A. Clark, and P. Rohatgi, "Policy evolution with grammatical evolution," in *Proceedings of the 7th International Conference on Simulated Evolution And Learning (SEAL '08)*, ser. Lecture Notes in Computer Science, X. Li, M. Kirley, M. Zhang, D. G. Green, V. Ciesielski, H. A. Abbass, Z. Michalewicz, T. Hendtlass, K. Deb, K. C. Tan, J. Branke, and Y. Shi, Eds., vol. 5361. Melbourne, Australia: Springer, Dec. 7-10 2008, pp. 71–80.
- [16] P. C. Cheng, P. Rohatgi, C. Keser, P. A. Karger, G. M. Wagner, and A. S. Reninger, "Fuzzy multi-level security: An experiment on quantified risk-adaptive access control," *sp*, vol. 0, pp. 222–230, 2007.
- [17] H. Inoue and S. Forrest, "Inferring java security policies through dynamic sandboxing," in *In International Conference on Programming Languages and Compilers, Las Vegas*, 2005.
- [18] D. Scott and R. Sharp, "SPECTRE: A tool for inferring, specifying and enforcing web-security policies," 2002.
- [19] P. Domingos and G. Hulten, "Mining high-speed data streams," in *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2000, pp. 71–80.
- [20] W. Fan, "Systematic data selection to mine concept-drifting data streams," in *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2004, pp. 128–137.
- [21] P.-C. Cheng and P. A. Karger, "Risk Modulating Factors in Risk-Based Access Control for Information in a MANET," IBM Research Report RC24442, Tech. Rep., 2008.
- [22] J. R. Koza, D. Andre, F. H. Bennett, and M. A. Keane, *Genetic Programming III: Darwinian Invention & Problem Solving*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.
- [23] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm," in *EUROGEN 2001. Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*, K. Giannakoglou, D. Tsahalis, J. Periaux, P. Papailou, and T. Fogarty, Eds., Athens, Greece, 2002, pp. 95–100.
- [24] S. Luke, "ECJ version 18 A Java-based Evolutionary Computation Research System," May 2008. [Online]. Available: <http://cs.gmu.edu/~eclab/projects/ecj/>
- [25] M. Hollander and D. A. Wolfe, *Nonparametric statistical inference*. New York: John Wiley & Sons, 1973.
- [26] A. Brindle, "Genetic algorithms for function optimization," Ph.D. dissertation, University of Alberta, Edmonton, Alberta, Canada, Jan. 1981, computer Science Department, Technical Report TR81-2.
- [27] S. Bleuler, M. Brack, L. Thiele, and E. Zitzler, "Multiobjective genetic programming: Reducing bloat using SPEA2," in *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*. COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea: IEEE Press, 27-30 May 2001, pp. 536–543. [Online]. Available: <ftp://ftp.tik.ee.ethz.ch/pub/people/zitzler/BBTZ2001b.ps.gz>
- [28] J. Branke, *Evolutionary Optimization in Dynamic Environments*. Norwell, MA, USA: Kluwer Academic Publishers, 2001.
- [29] A. Krogh and J. Vedelsby, "Neural network ensembles, cross validation, and active learning," in *Advances in Neural Information Processing Systems*. MIT Press, 1995, pp. 231–238.



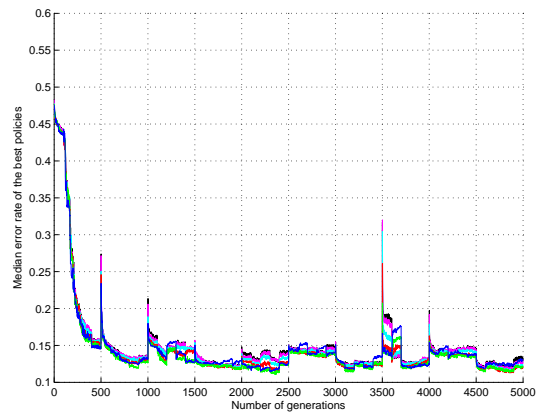
(a) Experiment 4: Unweighted voting mechanism



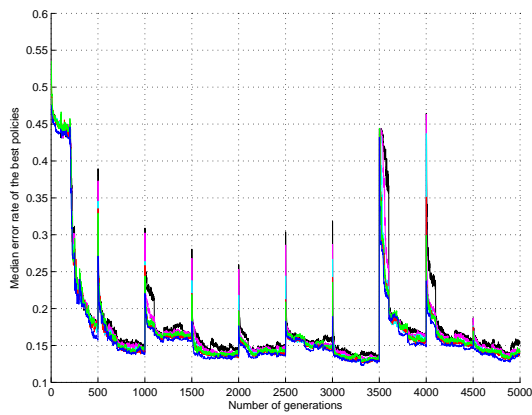
(b) Experiment 4: Weighted voting mechanism



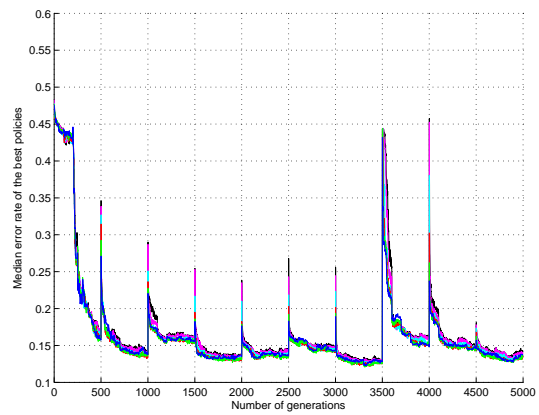
(c) Experiment 5: Unweighted voting mechanism



(d) Experiment 5: Weighted voting mechanism



(e) Experiment 6: Unweighted voting mechanism



(f) Experiment 6: Weighted voting mechanism

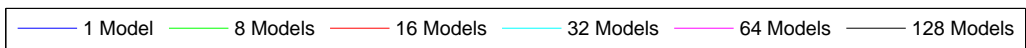


Fig. 9. Performance of Ensemble Models in Experiments 4, 5 and 6