Training Genetic Programming on Half a Million Patterns: An Example From Anomaly Detection

Dong Song, Malcolm I. Heywood, *Member, IEEE*, and A. Nur Zincir-Heywood, *Member, IEEE*

***Abstract*—The hierarchical RSS-DSS algorithm is introduced for dynamically filtering large datasets based on the concepts of training pattern age and difficulty, while utilizing a data structure to facilitate the efficient use of memory hierarchies. Such a scheme provides the basis for training genetic programming (GP) on a data set of half a million patterns in 15 min. The method is generic, thus, not specific to a particular GP structure, computing platform, or application context. The method is demonstrated on the real-world KDD-99 intrusion detection data set, resulting in solutions competitive with those identified in the original KDD-99 competition, while only using a fraction of the original features. Parameters of the RSS-DSS algorithm are demonstrated to be effective over a wide range of values. An analysis of different cost functions indicates that hierarchical fitness functions provide the most effective solutions.**

***Index Terms*—Dynamic subset selection (DSS), genetic programming (GP), hierarchical cost function, intrusion detection, large data sets.**

I. Introduction

THE Internet, as well as representing a revolution in the ability to exchange and communicate information, has also provided greater opportunity for disruption and sabotage of data previously considered secure. The study of systems for resisting such events—intrusion detection systems (IDSs)—naturally provides many challenges. In particular, the environment is forever changing, both with respect to what constitutes both normal and abnormal behavior. Moreover, given the widespread utilization of networked computing systems, it is also necessary for such detectors to provide very low false alarm rates in comparison to other classification type systems [1]. In order to promote the comparison of advanced research in this area, the Lincoln Laboratory at MIT, under the Defense Advanced Research Projects Agency (DARPA) sponsorship, constructed the 1998 and 1999 intrusion detection evaluations [1], [2]. As such, a basis for making comparisons of existing systems is provided under a common set of circumstances and assumptions. Based on binary transmission control protocol (TCP) dump data provided by the DARPA evaluation, millions of connection statistics were collected and generated to form the training and test data in the Classifier Learning Contest organized in conjunction with the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining 1999 (KDD-99) [3]. The learning problem is to build a detector (i.e., a classifier) capable of distinguishing between "bad" connections, called intrusions or attacks, and "good" or normal connections. There were a total of 24 entries submitted to the original contest [3], [4]. The top three winning solutions are all variants of decision trees. The winning entry is composed from 50×10 C5 decision trees fused by cost-sensitive bagged boosting [5]. The second placed entry consisted of a decision forest containing 755 trees [6]. The third placed entry consisted of two layers of voting decision trees augmented with human security expertise [7]. Each of these systems is designed to provide separate classifications for each type of attack—a requirement of the competition [1]—resulting in rather complex detector architectures which might not scale with time, i.e., attacks and attack types are in a continuous cycle of evolution. An alternative approach is that of anomaly detection [8]. In this case, there are only two classes, normal and anomalous, where the latter is composed from all behaviors judged to differ from the expected norm.

As a first step to developing solutions using genetic programming (GP), we are interested in the case of anomaly detection—multiple classifiers (detectors) would provide the basis for classification of different attacks should a multiclass solution be required. As a consequence, our principle interest lies in identifying a computationally efficient scheme for training GP on large data sets. Specifically, the KDD-99 benchmark consists of three data sets—whole, 10%, and corrected—where 10% KDD-99 is used for training and corrected KDD-99 is used for test. 10% KDD-99 comprised of half a million patterns over a dimension of 41 features. Previous work on GP has typically resorted to hardware specific approaches when "large" data sets were involved. Specific examples include the use of Beowulf computing clusters [9], parallel computers [10], reconfigurable computing platforms [11], or platform specific binary machine code implementation of GP [12]. The principle exception to this was the work of Gathercole on dynamic subset selection (DSS) [13]. In this case, a process is defined for subsampling the original training set based on the learning algorithm perception of training pattern difficulty and age. Such a scheme was demonstrated on a data set of 3772 patterns. In this paper, we build on the DSS paradigm to provide a hierarchy of subset selections compatible with the organization of memory hierarchies widely employed in computer architectures [14]. Each level of the subset hierarchy is, therefore, smaller than the previous and accessed more frequently. The only requirements of the ensuing

Manuscript received May 6, 2003; revised January 21, 2004. This work was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada under Discovery Grant 238791-01 and Discovery Grant 239138-01, and in part by the New Opportunities Project 7178 from the Canada Foundation for Innovation (CFI).

D. Song is with Quest Software, Inc., Halifax, NS B3J 1M5, Canada (e-mail: dsong@cs.dal.ca).

M. I. Heywood and A. N. Zincir-Heywood are with the Computer Science Department, Dalhousie University, Halifax, NS B3H 1W5, Canada (e-mail: mheywood@cs.dal.ca; zincir@cs.dal.ca).

Digital Object Identifier 10.1109/TEVC.2004.841683

algorithm—hereafter referred to as hierarchical RSS-DSS—are that the dataset be labeled and that the selection operator takes the form of a steady-state tournament.

A second objective of the work was to investigate the significance of utilizing anything other than the most basic KDD-99 features. That is to say, the KDD-99 benchmark has come under criticism for utilizing features that basically flag specific attack types. Moreover, KDD-99 provides an *a priori* set of temporal features in the feature vector. This work only utilizes eight of the 41 features, representing the most "basic" KDD-99 features, while requiring temporal and content-based characteristics to be derived by GP itself.

The ensuing system is demonstrated on a 1 GHz Pentium III laptop with 256 Mbyte RAM. Training times take on average 15 min to establish detection rates of $\approx$90% and false positive rates of less than 1.0%. Parameters of the RSS-DSS algorithm are qualified empirically, and the significance of different fitness functions investigated (classification count alone, weighted classification counts, and hierarchical cost). This is particularly significant in this application as the data set is strongly biased in favor of patterns representing normal and denial of service attacks. Other attack types with only a fraction of a percentage of instances in the training set might, therefore, be ignored entirely, significantly reducing the functionality of the overall system.

The paper continues by first discussing the characteristics of the KDD-99 data set in Section II. Section III summarizes the operation of the linear (page-based) GP employed, where the results reported are naturally not dependent on the particular structure of GP employed. The hierarchical RSS-DSS algorithm is detailed in Section IV. Preprocessing necessary to ensure the minimal *a priori* assumption to the construction of GP solutions is defined in Section V along with the rationale and definition of the various fitness functions. Parameterization of the algorithm and results detailing performance are given in Section VI with Section VII describing related works such as boosting and coevolution. Section VIII concludes the work. For completeness, Appendix A is included in which a GP solution is analyzed [15]. This demonstrates that GP has indeed produced a solution of interest to the wider intrusion detection literature.

II. KDD-99 Data Set

From the perspective of the GP paradigm the size of the KDD-99 data set is much larger than normally the case in GP applications. The entire training data set consists of about 5 000 000 connection records. However, KDD-99 provided a concise training data set—which is used in this work—and appears to be utilized in the case of the entries to the data-mining competition [3]–[7]. Known as "10% training," this contains 494 021 records—still a considerable challenge for GP—where a solution to this problem alone represents a major contribution of this work.

The KDD-99 data set describes each connection record in terms of 41 features and a label declaring the connection as either normal, or as a specific attack type. There are nine "intrinsic features of a single connection;" hereafter referred to as "basic features" [16]. The additional 32 *derived* features, fall into three categories.

TABLE I
Distribution of Attacks

Data Type	Training	Test
Normal	19.69%	19.48%
Probe	0.83%	1.34%
DOS	79.24%	73.90%
U2R	0.01%	0.07%
R2L	0.23%	5.2%

TABLE II
Distribution of Training and Test Data

Connection	Training	Test
Normal	97249	60577
Attacks	396744	250424

- **Content features**: Domain knowledge is used to assess the payload of the original TCP packets. This includes features such as the number of failed login attempts.
- **Time-based traffic features**: These features are designed to capture properties that mature over a 2-s temporal window. One example of such a feature would be the number of connections to the same host over the 2-s interval.
- **Host-based traffic features**: Utilize a historical window estimated over the number of connections—in this case 100—instead of time. Host-based features are therefore designed to assess attacks, which span intervals longer than 2 s.

In this work, none of these additional 32 features are employed, as they are derived from these nine basic features by expert domain knowledge, hence explicitly crafted to help distinguish suspicious connections [16]. Our interest is on assessing how far the GP paradigm would go on "basic features" alone. Moreover, the form of GP employed indexes features using an order 2 scheme, thus, the first eight features are utilized in this work, resulting in each connection being described in terms of: duration; protocol; service; normal or error status of the connection (Flag); number of data bytes from source to destination (DST); number of data bytes from destination to source (SRC); whether the destination and source addresses are the same (LAND); and the number of wrong fragments (WRONG).

The training data[1] encompasses 24 different attack types, grouped into one of four categories: user to root (U2R, unauthorized access to superuser (root) privileges, e.g., various "buffer overflow" attacks); remote to local (R2L, unauthorized access from a remote machine, e.g., guessing password); denial of service (DOS, e.g., syn flood); and probe (surveillance and other probing, e.g., port scanning) [2]. Naturally, the distribution of these attacks varies significantly, in line with their function—"DOS," for example, results in many more connections than "probe." Table I summarizes the distribution of attack types across the training data. Test data,[2] on the other hand, follows a different distribution, where this has previously been shown to be a significant factor in assessing generalization [3]. Moreover, the test data includes additional 14 attack types not present in the training data, and therefore considered as a

[1]"10% KDD-99 data set" in KDD-99 contest [3].

[2]"corrected test set" in KDD-99 contest [3].

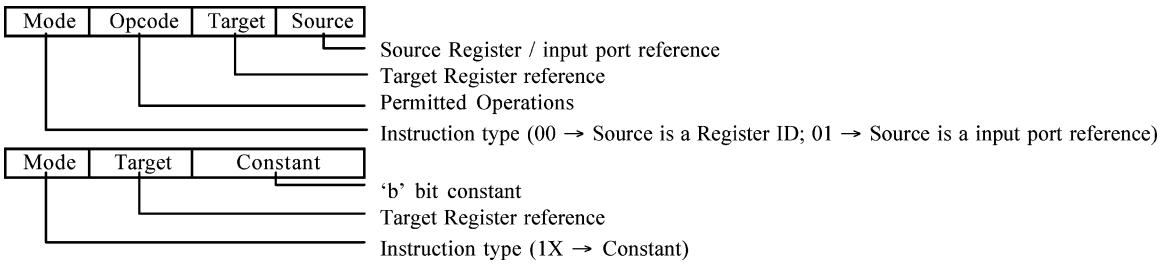


Fig. 1. Composition of a two-address instruction.

good test of generalization [3]. Finally, Table II summarizes the count of connections comprising the training and test sets.

III. Dynamic Page-Based Linear Genetic Programming (GP)

Linearly structured GP is based on a representation closely related to that employed by genetic algorithms. Specifically, individuals are constructed from a (linear) sequence of integers each of which has to be decoded into a valid instruction (syntactic closure). The decoding process effectively translates each integer into an equivalent binary string, separates the string into a series of fields based on the addressing mode, and maps each field into a valid value. Typical fields include mode, opcode, source, and destination (Fig. 1). The mode bit distinguishes between different instruction types, for example instructions detailing a constant or an operation performed on a register or on an input. The source and destination fields detail specific registers or input ports. Programs now take the form of a register level transfer language in which all operations operate on general purpose registers or read values from input ports (features from the current example). In this paper, a two-address instruction format is employed, e.g., $\mathrm{R1} \leftarrow \mathrm{R1} + \mathrm{IP3}$, where R1 denotes the first general purpose register and IP3 is a reference to the 3rd feature of the current example. The "opcode" may be considered equivalent to the concept of a functional set in tree structured GP, with the exception that constants are specified by the mode bit not through the opcode (Fig. 1).

As with the case of tree structured GP many instances of linear GP (L-GP) have been developed over a considerable period of time [12], [18], [20], [21]. The emphasis of this work, however, is the RSS-DSS algorithm, from which the only requirement is that the selection operator be a steady-state tournament. The specific form of Linearly structured GP (L-GP) employed by this work utilizes the page-based L-GP developed in an earlier work [22]. Such a scheme enforces a fixed length representation (crossover only exchanges an equal number of instructions), the basic components of which are defined as follows.

- *Representation*: Individuals take the form of a two-address instruction format (Fig. 1). Individuals are described in terms of a (uniform) randomly selected number of pages, where each page has the same fixed number of instructions. Instructions comprising an individual have no redundancy (all integers decode into a valid instruction [Fig. 1]).
- *Initialization*: Individuals are described in terms of the number of pages and instructions, where instructions are selected from a valid set of integers denoting the instruction set. The number of pages per individual is determined through uniform selection over the interval $[1, \ldots, \mathrm{maxPages}]$. That is to say the initial population is initialized over the entire range of program lengths. Defining an instruction is a two-stage process in which the mode bit is first defined (instruction type) using a roulette wheel (user specifies the proportions of the three instruction types). Second, the content of the remaining fields is completed with uniform probability. Such a scheme is necessary in order to avoid half of the instructions denoting constants, i.e., effect of the mode field (Fig. 1).
- *Selection operators*: The RSS-DSS algorithm requires a steady-state tournament. In this case, all such tournaments are conducted with four individuals selected from the population with uniform probability. The two fittest individuals are retained and reproduce. The children overwrite the worst two individuals from the same tournament using their respective position in the population.
- *Variation operators*: Three variation operators are utilized, each with a corresponding probability of application, where such tests are applied additively (i.e., the resulting children might be the result of all three variation operators). Crossover selects one page from each offspring and swaps them. The pages need not be aligned, but always consist of the same number of instructions. Mutation has two forms. The first case—hereafter referred to as "mutation"—merely Ex-ORs an instruction with a new instruction. No benefits were observed in making such a mutation operator "field specific," where this is undoubtedly a factor of the addressing format [22]. The second mutation operator—hereafter denoted "swap"—identifies two instructions with uniform probability in the same individual and interchanges them. The basic motivation being that an individual might possess the correct instruction mix, but have the instruction order incorrect.

This represents the basic page-based L-GP scheme. However, the selection of page size is problem specific. As a consequence, the dynamic page-based L-GP algorithm was introduced to modify the number of instructions per page dynamically during the course of the training cycle [22]. In this case, the user merely defines the maximum page size as an order of 2. The page size is then doubled for each plateau in the fitness function, beginning with a page size of 1 and finishing at "max page

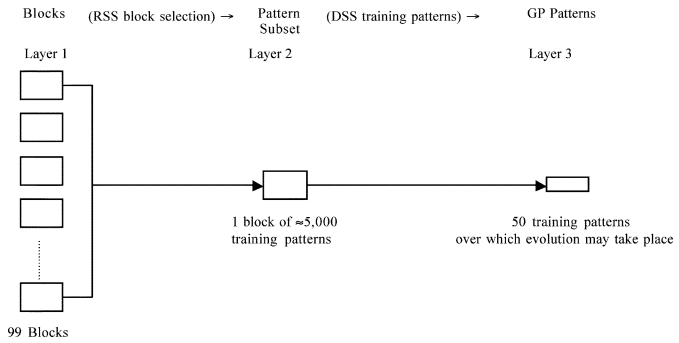


Fig. 2. RSS-DSS hierarchy.

size" and returning to a page size of 1 once a plateau following "max page size" is encountered. Plateaus are defined in terms of consecutive nonoverlapping windows of ten tournaments. For each of the ten tournaments, the (tournament's) best-case individual's fitness is summed. If the total over both windows is the same then a plateau is "defined." Such a scheme was demonstrated to be much more robust than that of a fixed page size over a range of benchmark problems (two boxes, six parity, UCI classification benchmarks) [22]. We emphasize, however, that the principle interest of this paper lies in the RSS-DSS algorithm, which is equally applicable to any form of GP so long as a steady-state tournament is employed for the selection operator. This constraint appears as training subsets identified by the hierarchical RSS-DSS algorithm are not evaluated across the entire population. Such a process although feasible, would result in *all* individuals being assessed on a subset of the entire training data, possibly resulting in degenerate solutions dominating the population. In order to emphasize the incremental nature by which the population is updated, the combination of steady state tournament and therefore replacement of the worst performing half of the tournament by the children from the better performing half is referred to as an "evolutionary phase."

IV. RSS-DSS Algorithm

The specific interest of this work lies in identifying a solution to the problem of efficiently training GP with a large data set (close to half a million patterns in the case of KDD-99). There are at least two aspects to this problem: the cost of fitness evaluation—the inner loop, which dominates the computational overheads associated with applying GP in practice; and the overhead associated with managing data sets that do not fit within RAM alone. In this work, the basic approach is based on the concept of a hierarchy in which the data set is first partitioned into blocks (Fig. 2). The blocks are small enough for retention in RAM where the result of a competition is a subset of training patterns that reside within cache alone. A set number of competitions take place between training patterns within a selected block. The selection of blocks is performed using random subset selection (RSS)—layer 1 in the hierarchy (Fig. 2). DSS enforces a competition between connections within a block—layer 2 in the hierarchy (Fig. 2). Thus, a hierarchical architecture has been defined in conjunction with the previous concepts of RSS and DSS in order to facilitate access to a much larger training set. RSS and DSS were previously only demonstrated in isolation on a data set of 3772 patterns, therefore, never requiring the concept of a hierarchy [13].

The generic hierarchical RSS-DSS algorithm consists of five basic stages (Fig. 3).

1) Division of the dataset into blocks of sufficiently small size to retain a block within memory—step 1, Fig. 3: Each block is of the same size, thus, for a total of P training patterns and a block size of B there are P/B blocks, layer 1, Fig. 2.
2) Selecting a block with uniform probability, or RSS—step 2.a, Fig. 3: Implies that all blocks are treated equally for the duration of the training cycle, layer 2, Fig. 2.
3) Selecting a subset of the training patterns from the block (with replacement), or DSS—step 2.b.i, Fig. 3: Subset selection is performed in proportion to a predefined ratio of age and difficulty (experimented empirically in Section VI) and the relative age and difficulty of patterns with respect to each other. The implication being that two "roulette wheels" exists, one for age and one for difficulty, layer 3, Fig. 2.
4) Training the learning algorithm on a subset—step 2.b.ii, Fig. 3: At this point, the current training subset defines the "data set" over which an "evolutionary phase" takes place. By using an evolutionary phase, we imply that following fitness evaluation—performed over the members of the steady state tournament alone—the better performing half of the tournament denote the parents. Application of the variation operators creates the children that then overwrite the respective worst members of the tournament, that is in the population itself. The critical point here is that the process is limited to the subset of individuals first identified by the steady state tournament. The number of such evolutionary phases performed is defined by the number of DSS iterations—step 2.b, Fig. 3—whereas the training subset used to evaluate fitness is resampled modulo "*DSS-RefeshFreq*"—step 2.b.i, Fig. 3.
5) Modifying the number of DSS subsets performed on the block—step 2.e, Fig. 3. This is the principle mechanism for decreasing the number of patterns over which GP iterates. That is to say, initially the hierarchical RSS-DSS algorithm is parameterized such that the number of subsets sampled matches the number of patterns per block. (Further empirical evidence for the selection of block size and subset samples per block is given in Section VI-A.) As the performance of GP improves, then the number of subsets per block decreases. Naturally, the performance of GP will be block specific, resulting in a block specific number of subset samples or DSS iterations per block $(DSSiteration.\mathrm{B})$.

In addition, a series of supporting design decisions are necessary.

- Frequency with which subset content is refreshed—step 2.b.i, Fig. 3: The use of a steady-state tournament implies that only a small number of individuals from the GP population are trained on a subset at a time. Thus, multiple evolutionary phases may take place on the same subset without penalty. This is termed the (DSS) refresh

1. Define Blocks by dividing the original training set into P/B equal partitions;
2. FOR i=0; i<*maxRSSiteration*; i++
   a. With uniform probability select block 'B'; ∀ patterns ∈ B, B.pattern{age, difficulty} = 0.
   b. FOR j=0; j<*DSSiteration*.B; j++
      i. IF (j mod (*DSSRefreshFrequency*) == 0) THEN
         1. FOR k=0; k<*subsetSize*; k++
            IF rand < P(age)
            THEN (select PatternSubset(k) ∝ B.pattern.age)
            ELSE (select PatternSubset(k) ∝ B.pattern.difficulty);
         2. ∀ pattern ∉ Subset, B.pattern.age++;
      ii. Perform an 'evolutionary phase' over patterns defined by PatternSubset;
         1. Choose steady state tournament individuals with uniform probability from population;
         2. Evaluate fitness for subset of individuals over PatternSubset;
         3. Apply variation operators to the selected subset of individuals;
         4. Update population with children;
         5. B.pattern.difficulty += dist(individual, B.pattern);
   c. LET bestGPindividual = $\text{Argmin}_{\text{individual} \in \text{DSSiteration.B}}$(fitness);
   d. FOR m=0; m<patterns ∈ B; m++
      i. Error(B)+= dist(bestGPindividual, B.pattern);
   e. *DSSiteration*.B = f(*maxDSSiteration*, Error (B));

Fig. 3. Generic RSS-DSS algorithm.

frequency ($DSSRefreshFrequency$). A refresh frequency of 1 naturally implies that the subset content changes for every evolutionary phase. In this work, the refresh frequency is set at 20% of the population size (or six tournaments for a population size of 125).

- Protocol for updating age—step 2.b.i.2, Fig. 3: Age is updated after the content of a subset is refreshed. Note, we do not reset the age of training patterns participating within a subset to zero, but merely skip incrementing their respective ages. The motivation being that resetting ages to zero would effectively completely preclude reselection within the current block visit (large difference in age between patterns previously selected and those not previously selected).
- Protocol for updating pattern difficulty—step 2.b.ii.5, Fig. 3: Pattern difficulty is updated with respect to each GP individual participating in an evolutionary phase using a suitable distance metric. In the case of classification problems, this merely reflects whether the training pattern was correctly classified or not (distances of 0 and 1, respectively).
- Function for updating the number of DSS subsets sampled per block—step 2.e, Fig. 3: All blocks have an initial maximum number of DSS subsets (discussed Section VI-A). Let this be "$maxDSSiteration$." Thereafter, the number of DSS iterations per block decreases in proportion to GP performance measured in terms of a normalized fitness function, i.e., zero represents a perfect solution and unity the worst possible solution. To this end the following expression is employed for updating the number of DSS (subset) iterations on block "B"

$$DSSiteration.\mathrm{B} = maxDSSiteration \times Error(\mathrm{B})$$

where $DSSiteration.\mathrm{B}$ is the number of DSS subsets selected on block "B" at the next selection of the block; and $Error(\mathrm{B})$ is the number of misclassifications of the best individual taken from the last evolutionary phase on block "B." Hence, $Error(\mathrm{B}) = 1 - [\mathrm{hits(B)}/\#\mathrm{pattern.B}]$, where hits(B) is the hit count over all training patterns from block "B" for the best case individual taken from the last tournament on block "B" and #pattern.B is the total number of training patterns in block "B."

V. Representation of Time and the Definition of Fitness Functions

A. Representation of Time

Constructing detectors on the basic KDD-99 features alone implies that the learning algorithm is required to determine any necessary temporal features itself. Previous experience with machine learning approaches to the IDS problem has indicated

that only the relative sequence between connections is important and not the absolute time stamp [16]. To this end, the following shift register structure is utilized. For each "current" connection record $x(t)$ GP is permitted to index the previous 28 connection records relative to the current sample t, modulo 4, the "tap." Thus, for each of the eight basic features available in the KDD-99 data set, GP may index a total of eight connection records $[(t), (t-4), \ldots, (t-28)]$, where the objective is to provide the label associated with sample "t." Thus, each new incoming connection propagates the contents of the shift register one location to the right, and the last entry "rolls off" the end of the shift register. This presents an input space of 8×8 (64) locations. Section VI-D reports on additional experiments with variations in tap distance and, therefore, the significance of shift register resolution and depth.

B. Fitness Functions

This work considers three different fitness functions. The principle motivation for doing so is in recognition of the unequal distribution of different attack classes over the training set, Table I; this has several implications. Not only are specific attack classes more frequent than others, but the most frequent classes do not necessarily have any correlation with the classes which are more difficult to classify. For example, DOS connections, by the very nature of the attack type, have a high frequency of occurrence, but conceptually at least are relatively easy to identify. On the other hand, the "probe" attack class is relatively infrequent and represent a much less obvious form of intrusion, Table I. Thus, high classification rates are possible on training data (>99%) if the detector ignores all class types other than "normal" and "DOS." However, such a system would not be deemed usable in practice. This would be reflected in a poor false positive rate—where rates smaller than 1% are necessary for a system to be of any practical value—and a poor test set classification accuracy as the distribution of remote-to-local (R2L) attacks increases significantly between training and test sets, Table I. Thus, although the detector being developed is only required to produce a binary classification (normal or not normal) it is anticipated that the fitness function should reflect the wider context of the classification problem.

With this in mind, three fitness function are considered.

- **Equal class cost**: This represents the baseline case in which a standard hits-based cost function is employed $E = \sum_{i=1}^{\mathrm{NSS}} \overline{\mathrm{hit}}$, where $\overline{\mathrm{hit}}$ is 0 if wrapper output of GP individual is correct and 1, otherwise, and NSS is the number of connection records in the subset.
- **Variable class cost**: In an attempt to "reward" the classification of infrequent attack types, these cases receive a higher or weighted payoff. Moreover, as the detector becomes more adept at classifying specific classes, the class weights should be reweighted to reflect this. There are, therefore, two properties of interest. The initial distribution of weights associated with each category and the rate at which dynamic reweighting occurs. Specifically, at evolutionary phase t, the weight on category c, $w^c(t)$ is calculated as an exponential weighted history of previous weight values

$$w^c(t) = \alpha \times E_b^c(t-1) + (1-\alpha) \times w^c(t-1)$$

where $E_b^c(t-1)$ is the block b misclassification count on category c at the previous iteration and $0 < \alpha < 1$. Naturally, depending on the value for the initial class weights $w^c(0)$ and the rate of change parameter α significantly different detectors will be evolved. Specific parameter selections and rationale are discussed in Section VI-A.
- **Hierarchical cost**: Recently, success has been reported when using GP in conjunction with a hierarchical or lexicographic cost function [18], [19]. However, both the earlier works utilized a hierarchical cost function to resolve the trade off between error and complexity of GP individuals. In the case of this work, a hierarchical formulation is used to decrease the number of parameters required to configure the cost function, while still "weighting" different classification categories. In essence, a hierarchical formulation provides a series of cost functions such that when a tie appears at cost function i, then cost function $i+1$ is utilized to resolve the ranking. In this case, two cost functions are employed. The level 1 cost function measures misclassification rate over "normal" and "DOS;" that is to say without first establishing sufficient accuracy on classes representing 99% of the training set, there is little point in recording that over the remaining 1%. Level 2 expresses the misclassification rate over the remaining three attack categories—probe, user-to-root (U2R), and remote-to-local (R2L)—Table I. Misclassification is, therefore, expressed over specific classes and, in the case of level 1, rounded to the nearest integer. This latter requirement enforces a tolerance over which misclassification rates are judged equal. Without this, the diversity of patterns at level 1 would be sufficient to render level 2 ineffective. Thus, the level 1 and 2 costs are, respectively

$$\mathrm{round}\left(100 \times \frac{\sum_{p=1}^{P'} \overline{\mathrm{hit}}(p)}{\#P'}\right)$$

where P' is the number of training patterns in the DOS and Normal subset and "round" returns the nearest integer value and

$$\sum_{p=1}^{P''} \overline{\mathrm{hit}}(p)$$

where P″ is the number of training patterns in the remaining categories.

VI. Results

A. Parameterization of RSS-DSS Algorithm

From Section IV, Fig. 3, it is apparent that there are four basic parameters defining the hierarchical RSS-DSS algorithm: number of block selections $(maxRSSiteration)$; number

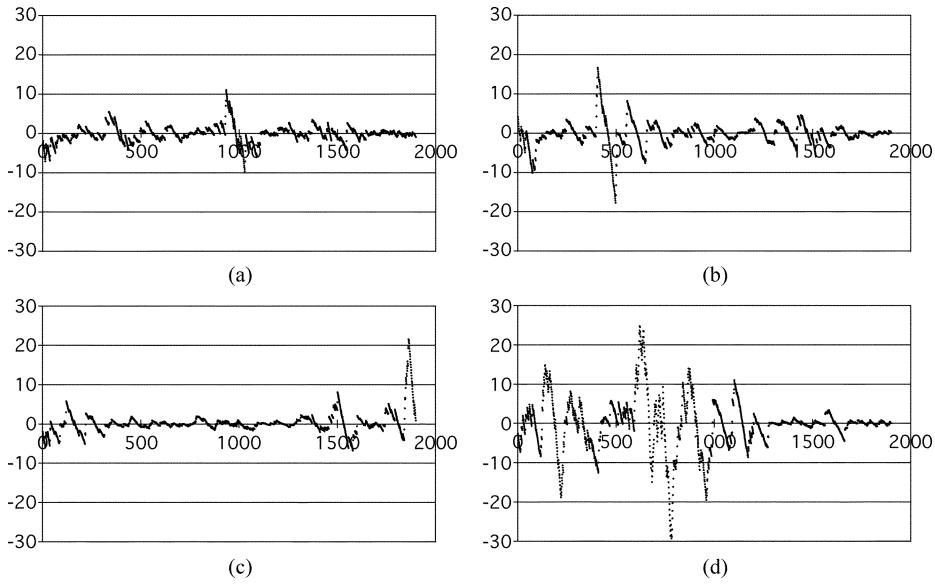


Fig. 4. Gradient of block error using best case DSS individual. x axis represents tournament and y axis represents error gradient over a sliding window of 100. (a) max DSS iteration of 100. (b) max DSS iteration of 200. (c) max DSS iteration of 400. (d) max DSS iteration of 800.

of evolutionary phases per block ($maxDSSiteration$); frequency with which the training subset on which tournaments were performed is resampled ($DSSRefreshFrequency$); and the number of training patterns in a subset ($subsetSize$). The later denotes the number of patterns over which the fitness of GP individuals is explicitly evaluated, i.e., the inner loop. Standard GP benchmarks might use anywhere between 10–50 patterns. In this work, the subset size is fixed at 50 patterns. DSS refresh frequency may take a value between unity (subset patterns refreshed every evolutionary phase) to "max DSS iteration" (all evolutionary phases see the same subset selection). As steady-state tournaments are employed, the same subset content can be retained, with a low likelihood of the same individual seeing the same subset, providing that the refresh frequency is expressed in terms of a sufficiently small percentage of the individuals in the population. All the results here utilize a refresh frequency of 20% of the population (i.e., six evolutionary phases per subset selection).

The total number of samples taken from any given block at initialization is denoted by $maxDSSiterations/DSSrefreshfrequency$. Having fixed "DSS refresh frequency" the most significant parameter is now "max DSS iterations." Setting such a parameter too high would naturally result in the population of individuals overlearning on the content of a specific block. A process which might take several block selections to "reverse." Setting "max DSS iterations" too low would reduce the efficiency of the overall algorithm. That is to say, having incurred the overhead of transferring a block to memory only a small number of patterns are sampled before the block is replaced. Using the number of patterns per block as the natural overlearning limit, we define the upper limit for the number of subset selections per block ($maxDSSiterations$) by

$$\#\text{patterns.B} \leq maxDSSiterations/DSSrefreshfrequency \times subsetSize \quad (1)$$

whereas the lower limit is set by a desire to maximize the number of subset selections per block, or,

$$\#\text{ patterns.B} \geq maxDSSiteration \times subsetSize \quad (2)$$

where # patterns.B is the number of patterns in block B.

Naturally, these limits are conservative and will be discounted by the adaptation of "Max DSS iterations" in proportion to the previous performance on the same block, step 2.e, Fig. 3. Thus, by fixing "DSS refresh frequency" in terms of population size and acknowledging that the number of training patterns per block is parameterized by the memory hierarchy, we have identified the range of valid values for "max DSS iterations."

In order to demonstrate these relationships more explicitly, we conduct a series of runs for different parameterizations of "max DSS iterations:" 100, 200, 400, 800. The first is on the lower limit and the last exceeds the upper limit. Results are expressed in terms of the slope of the interblock error. That is to say, a sliding window is constructed consisting of 100 block selection errors, and a linear least-squares regression performed. The slope of each linear regression is then plotted (Fig. 4), for a stop criteria (maxRSSiterations) of 2000 blocks. From Fig. 4, two basic observations are apparent. First, exceeding the upper limit (maxRSSiterations of 800), (1), results in an increase in the magnitude of the slope, implying that overlearning of specific blocks has taken place. Second, the remaining three parameterizations are all equally effective at minimizing "error slope." Moreover, curtailing the stop criteria (maxRSSiterations) to 1000 block selections was sufficient to establish the fitness plateau beyond which very little development of the best case individual's fitness took place. Additional experiments with a block size of 2500 reinforced these conclusions; the only measurable difference being a marginally faster evolutionary cycle as quantified by CPU time (smaller block transfer overhead).

TABLE III
COMMON GP PARAMETERS

Parameter	Setting
Population Size	125
Maximum number of pages	32 pages
Page size	8 instructions
Maximum working page size	8 instructions
Crossover probability	0.9
Mutation probability	0.5
Swap probability	0.9
Tournament size	4
Number of registers	8
Instruction type 1 probability	0.5 / 5.5 or 9%
Instruction type 2 probability	4 / 5.5 or 73%
Instruction type 3 probability	1 / 5.5 or 18%
Function set	{+, -, *, /}
Terminal set	{0, .., 255} $\cup$ {i_0, .., i_{63}}
RSS subset size	5000
DSS subset size	50
RSS iteration	1000
DSS iteration (6 tournaments/ iteration)	100
Wrapper function	0 if output <=0, otherwise 1

TABLE IV
PARAMETERIZATION OF VARIABLE COST FUNCTION

Case	Description
1	$\alpha = 0.3$, $w^c(0) = 0.2$, $c = \{1 \ldots 5\}$
2	$\alpha = 0.05$, $w^c(0) = 0.2$, $c = \{1 \ldots 5\}$
3	$\alpha = 0.3$, $w^{normal}(0) = 0.6$ and $w^{c \neq normal}(0) = 0.1$

B. Parameterization of GP

All experiments are based on 40 runs of dynamic page-based L-GP under the same partitioning of the training data into a linear sequence of 100 blocks (Fig. 2). Runs differ only in the choice of a random seed initializing the initial population. The same genetic page-based L-GP parameters are employed as in a previous study [22]. The functional set is selected with a bias toward simplicity, i.e., arithmetic operators alone. Table III lists the common parameter settings for all runs. The total numbers of training and test set patterns are summarized in Table II. In the case of the three fitness function formulations, Section V-B, all parameters other than that specific to the fitness function remain unchanged.

In the case of the variable class cost fitness function, two parameter choices are necessary: initial class weights $w^c(0)$, and rate of change parameter α. To this end, three scenarios are considered (Table IV). In cases 1 and 2, all initial class weights are considered equally important, or $w^c(0) = 1/\#\text{categories}$ and experimented with "low" and "high" values for rate of change, α. Case 3 biased the initial class weights in favor of "normal," while utilizing the larger rate of change.

In total over the five experiments—equal class cost, three parameterizations of the variable class cost, and hierarchical cost—200 GP runs are required, where the average computational running time is 15 min per run. That is to say, no run correctly classified all 1/2 million patterns in the training set, thus, there is very little variation in the time to complete a run. All runs are performed on a Pentium III 1 GHz laptop platform with a 256 Mbyte RAM under Windows 2000, where GP is implemented in C++ using the Borland 5.01 compiler.

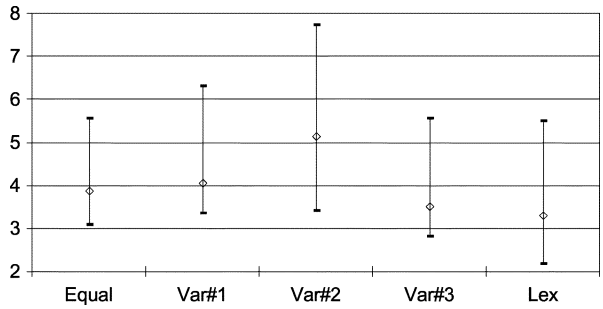


Fig. 5. Different cost functions: normal % misclassification error on test data—first, second and third quartile.

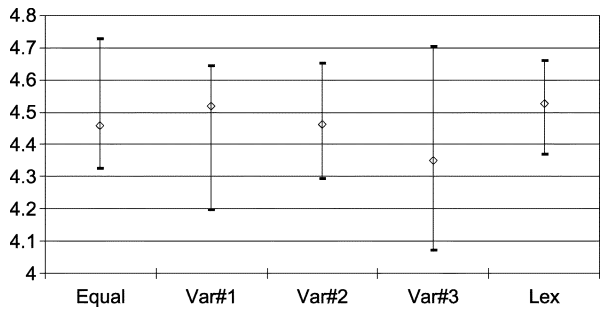


Fig. 6. Different cost functions: DOS % misclassification error on test data—first, second and third quartile.

The 40 best individuals within the last tournament are recorded and simplified. Note that "best" is defined with respect to the cost function used during training. Simplification takes the form of removing code that does not impact the eventual output of the program and is, therefore, only performed *post* training. Finally, the performance of the 40 "best" case solutions for each of the five experiments is expressed in terms of the typical IDS metrics of false positive (FP) and detection rate, estimated as follows:

$$\text{Detection Rate} = 1 - \frac{\#\text{False Negative Classifications}}{\text{Total Number of Attack Connections}}$$

$$\text{False Positive Rate} = \frac{\#\text{False Positive Classifications}}{\text{Total Number of Normal Connections}}.$$

C. Comparing Fitness Functions

Figs. 5–9 summarize the percentage of misclassification error on test data for each of the three fitness functions on a class-by-class basis; where positive classification corresponds to the normal class. In each case, results are expressed in terms of the first, second (median), and third quartiles over all 40 best-case solutions from each parameterization. Although no statistically significant distinction is expressed by a T-test at the 95% confidence interval, various trends are apparent. In particular, all cost functions perform well on the two most frequent classes—normal and DOS—Figs. 5 and 6. Probe represents the next best classification class (Fig. 7), where this also corresponds to the next largest category in training

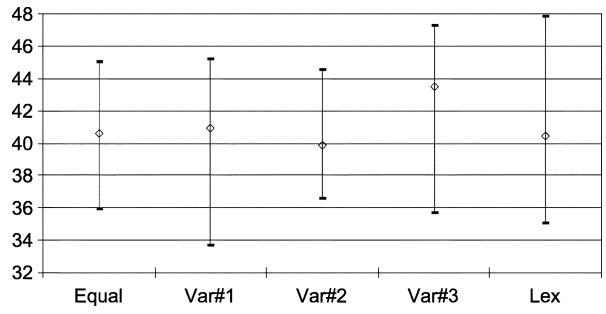


Fig. 7. Different cost functions: probe % misclassification error on test data—first, second and third quartile.

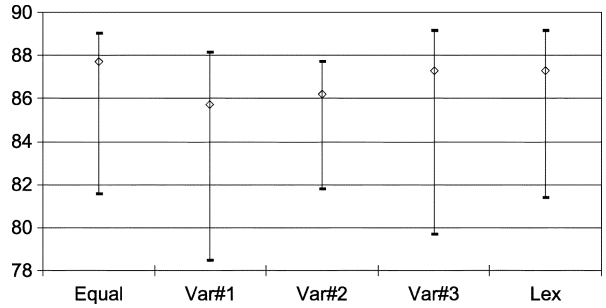


Fig. 8. Different cost functions: U2R % misclassification error on test data—first, second and third quartile.

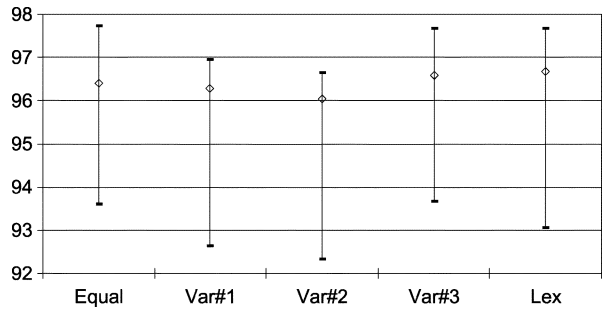


Fig. 9. Different cost functions: R2L % misclassification error on test data—first, second and third quartile.

at 0.83% of the total training data (Table I). Other than case 3 of the variable cost parameterization, all fitness functions return median misclassification rates of 40%, indicating a relatively good generalization from the small number of patterns available in the training data. The final two classes—R2L and U2R—have the least representation over the training data while also denoting content-based attacks. Thus, there is no direct way for the detector to recognize this form of attack (none of the content-based features from the original 41 features are supported). In spite of this, typically, 12% of U2R and 3% of R2L attacks are recognized (Figs. 8 and 9).

Solution complexity is expressed in terms of the number of instructions retained following simplification (Fig. 10). It is now apparent that solutions located using a hierarchical cost function are typically more complex. This is significant at the 90% confidence interval with respect to individuals trained using an equal class cost, and significant at the 95% confidence interval with respect to individuals trained using case 1 of the variable class cost. Moreover, if the best-case individuals for each cost

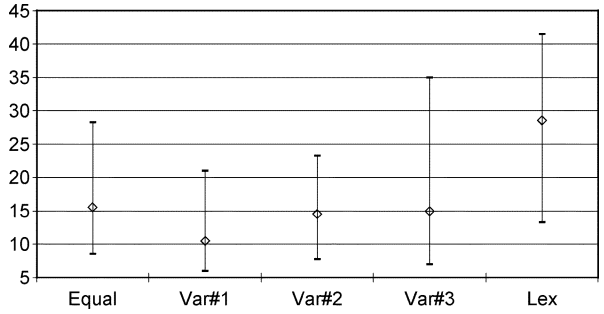


Fig. 10. Different cost functions: solution complexity after simplification—first, second and third quartile (y axis denotes instruction count).

TABLE V
COMPLEXITY AND CORRECTED (TEST) KDD-99 DATASET PERFORMANCE OF BEST CASE INDIVIDUAL: EQUAL, VARIABLE WEIGHT, AND HIERARCHICAL FITNESS FUNCTIONS WITH EIGHT FEATURES

Individual	Instruction Count	False Positive Rate (%)	Detection Rate (%)
Selection with respect to Best False Positive Rate			
Equal	64	0.6818	89.4096
Var#1	15	1.4461	88.9991
Var#2	21	1.6425	89.0334
Var#3	108	1.3570	88.8337
Selection with respect to Best Detection Rate			
Equal	7	3.2669	90.8252
Var#1	7	3.4112	89.9155
Var#2	6	4.8467	90.5037
Var#3	6	3.7539	90.3588
Hierarchical	86	0.9030	90.0233

function are identified with respect to test set detection and false positive rates, then a distinct pattern appears, Table V. Specifically, individuals identified as best-case detectors are always much more concise than those returning best-case false positives (the generality-specificity tradeoff). However, in the case of individuals trained under a hierarchical cost, the same individual provided both best-case detection and false positive rates. This theme is continued in Appendix A, where a "simple" individual provided under the equal class cost fitness function, and selected as an example of best case detection rate, is explicitly decoded and demonstrated to conform to various proposals from the IDS community [15].

D. Feature Selection and Difficulty-Age Weighting

The above experiments resulted in a preference for a hierarchical cost function. However, as indicated in Section IV, all the above experiments were performed with the first eight "intrinsic" connection features over a shift register of depth 28 with taps at every fourth position. Here, we qualify the degree of sensitivity of these results relative to the shift register structure. Moreover, experiments are also conducted against solutions evolved using all 41 KDD-99 connection features (shift register no longer necessary). Finally, results are also reported for a different weighting of difficulty and age (70/30 used above).

Two alternative formulations for the shift register are considered. A shift register spanning a deeper history of 48 connections by doubling the tap distance to 8. The second case also

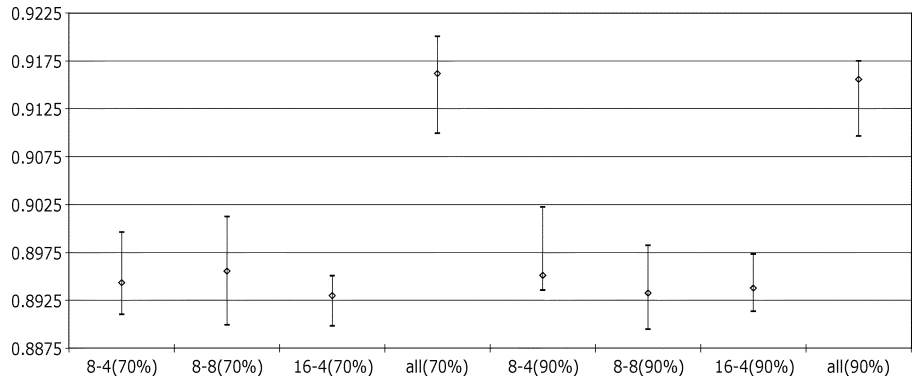


Fig. 11. Feature and age-difficulty ratio: detection rate—first, second and third quartile.

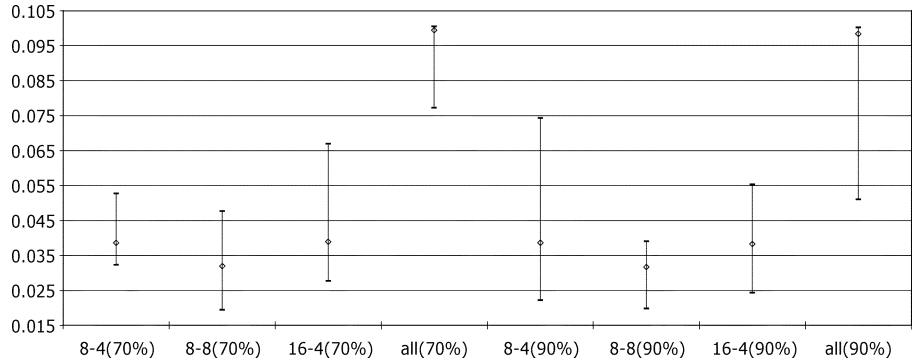


Fig. 12. Feature and age-difficulty ratio: false positive rate—first, second and third quartile.

utilizes a shift register depth of 48, but employs a finer tap resolution of 4. These are referred to as (8–8) and (16–4), respectively, with the original scheme denoted as (8–4). All parameters for hierarchical RSS-DSS and GP remain unchanged for both the 41-feature experiments and the additional shift register configurations. A hierarchical cost function is retained throughout and 30 runs performed for each input configuration.

Figs. 11 and 12 compare detection and FP rates under each input configuration and difficulty/age ratio in terms of first, second (median), and third quartiles. The most immediate difference lies in the case of solutions derived from all 41 KDD-99 features versus those based on the eight basic features. Trials based on all 41 features (all) unexpectedly always result in higher detection and FP rates (desirable and undesirable respectively), where this may be a factor of the instruction set (arithmetic operators not necessarily being effective at associating particular features with particular attacks). Second, we note that solutions based on the 70/30 ratio of difficulty and age appeared to result in tighter performance distributions than those at the 90/10 ratio, where this is apparent for both input configurations (especially, in the case of "all" and "8–4"). However, no significant difference in a (pairwise) comparison between medians was demonstrated. Thus, the addition of "age" appears to provide some additional consistency to the solutions located, independently of the features utilized. In the case of the solutions based on the eight basic features, no significant difference is observed between different shift register configurations. Fig. 13 details the distribution of solution instruction counts before simplification. No particular pattern is evident with respect to input configuration or difficulty-age ratio.

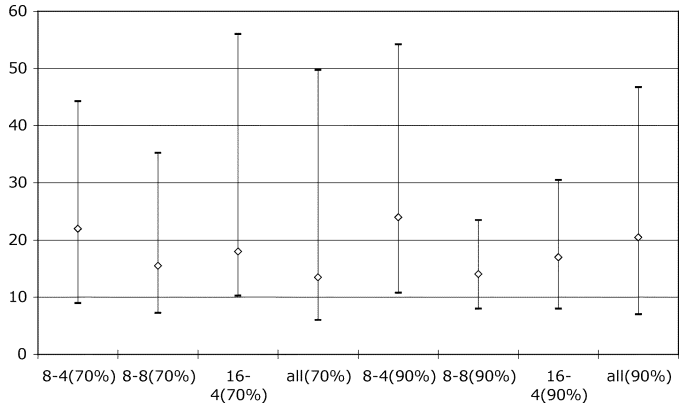


Fig. 13. Feature and age-difficulty ratio: solution complexity before simplification—first, second and third quartile (y axis denotes instruction count).

Figs. 14–18 detail the category specific error rates using first, second (median), and third quartiles. The principle difference is again between 41-feature (full) and 8-feature (basic) scenarios, where this is significant at the 99% confidence interval. Detection of normal is always worst under 41 features, whereas error rates under the four remaining categories is always much better. Smaller error rates are always provided for the three larger categories—representing 98.9% (93.38%) of the training (test) data—with U2R and R2L denoting the worst performing categories—corresponding to 0.24% (5.27%) of training (test)

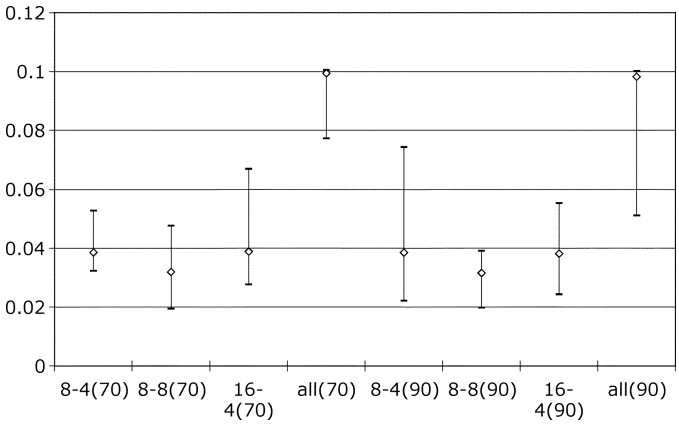


Fig. 14. Feature and age-difficulty ratio: normal % misclassification error on test data—first, second and third quartile.

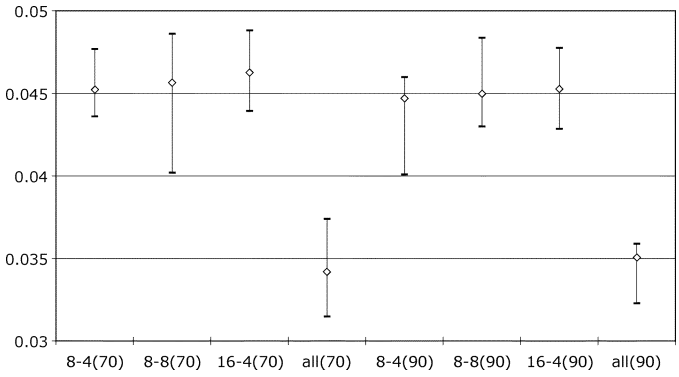


Fig. 15. Feature and age-difficulty ratio: DOS % misclassification error on test data—first, second and third quartile.

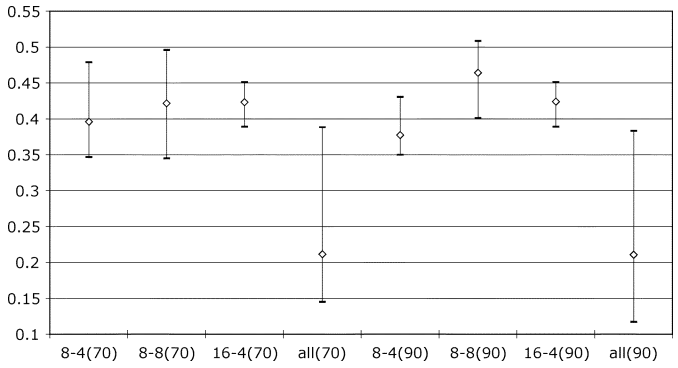


Fig. 16. Feature and age-difficulty ratio: probe % misclassification error on test data—first, second and third quartile.

data. Moreover, U2R and R2L tend to represent "content-based attacks," where the 41-feature scenario explicitly provides content-based features. With regards to the significance of different age-difficulty ratios, this appears to have the most impact in the case of the 41-feature scenario for categories of U2R, R2L, and normal. In each case, the higher weight of 90% results in a much wider distribution of results, without any improvement in the median. With respect to the eight-feature scenario, the combination of small tap and deep shift register history (16–4) does not appear to provide any advantage.

In order to summarize category specific performance and provide the basis for comparison against the KDD-99 competition

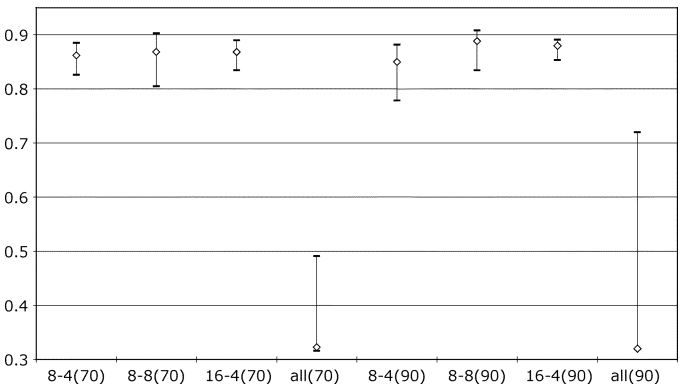


Fig. 17. Feature and age-difficulty ratio: U2R % misclassification error on test data—first, second and third quartile.

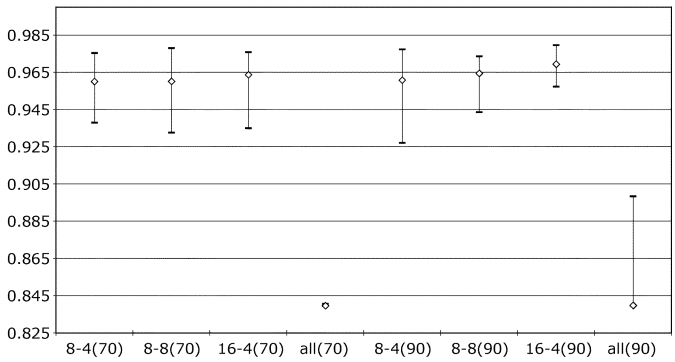


Fig. 18. Feature and age-difficulty ratio: R2L % misclassification error on test data—first, second and third quartile.

entries, we select one representative solution from each input configuration. To this end, the following metric is employed in which detection and FP rates are weighted equally

$$\frac{(1 - \text{FP rate} + \text{Detection rate})}{2}.$$

Table VI details the respective category specific error rates on KDD-99 corrected test dataset. As with the detection and FP rates, the detectors based an all 41 features emphasize maximizing detection, whereas the detectors based on eight features emphasize minimizing FP rate. The detector performance closest to that of the KDD-99 competition winners was the 8–4 and 8–8 detector at a 70% age weighting. The "all" feature GP detectors provided similar category specific performance to the original KDD-99 competition winners, but with a much higher FP rate. Moreover, it is also apparent that lowest FP rates are dominated by the performance under the larger categories of normal and DoS, whereas performance under the remaining smaller categories is dominated by detectors with better detection rates [e.g., 8–8 (70)].

VII. Related Work

The principle motivation for this work was to provide a framework for sampling the original training set such that the inner loop of GP need not iterate over the entire dataset. The basic observation supporting this is that from the perspective of the learning algorithm, not all training set patterns are equally significant, thus, the performance of a candidate solution may be

TABLE VI
CORRECTED (TEST) KDD-99 DATASET PERFORMANCE FOR CATEGORY SPECIFIC CLASSIFICATION, DETECTION, AND FP RATES OF BEST CASE INDIVIDUALS: HIERARCHICAL FITNESS FUNCTION WITH 8 AND 41 FEATURES

KDD 99 Competition Winners							
Model	Normal	DOS	Probe	U2R	R2L	Detect	FP
[5]	99.5	97.1	83.3	13.2	8.4	90.9	0.45
[6]	99.4	97.47	84.5	11.54	7.32	91.5	0.58
(full) 41 features							
70% diff.	96.5	99.7	86.8	76.3	12.35	94.4	3.5
90% diff.	98.6	96.7	85.7	59.2	9.3	91.4	1.3
8 basic features – Difficulty 70%; Age 30%							
8-4	99.1	95.36	62.6	9.2	1.6	89.27	0.92
8-8	99.0	96.1	81.0	60.5	1.6	90.3	1.00
16-4	98.6	95.5	56.5	11.4	0.8	89.26	1.35
8 basic features – Difficulty 90%; Age 10%							
8-4	99.7	95.6	48.5	10.1	0.2	89.2	0.27
8-8	98.0	95.6	55.4	18.0	3.4	89.47	1.9
16-4	98.7	95.7	55.1	10.2	1.8	89.49	1.7

used to filter the original dataset. Moreover, the problem is also formulated as a hierarchy of sub-samples in order to make use of the memory hierarchy widely supported on modern computing platforms. This is not, however, the first time that some of these observations have been made. In particular, ensemble classifiers have made widespread use of the observation that not all patterns are created equal in order to improve classifier accuracy. Indeed this observation is central to the approach taken by the winners of the KDD-99 competition [5]–[7]. Such schemes are typically based on bagging [23], boosting [24], or some combination of the two (bagged-boosting). Bagging algorithms sample the original training set size P, to produce "p" training set "instances," each of size P. Sampling is performed with replacement, using a uniform probability distribution. No information is utilized to bias the selection of each instance of the original training set. Each training set instance is used to train a new model under a suitable supervised learning scheme. The result is an ensemble of "p" models, each contributing an equal "vote." Such a scheme has been shown to require "unstable" learning algorithms in order to produce a sufficiently varied ensemble [23]. Boosting algorithms on the other hand utilize a "difficulty" measure to weight patterns from the training set. Training pattern weights are updated by each model added to the ensemble and are used to weight the cost function (boosting by weighting), the implication being that each instance of the training dataset is the same. A variation on this scheme uses the weights to provide a "roulette wheel" from which the original dataset is sampled (with replacement), creating a training set (also size P) with a distribution of training patterns in proportion to pattern difficulty (boosting by sampling). The basic update rule for "difficulty" in this case is a multiplicatively weighted function of learning history. Thus, in all cases there is no concept of "age" with which patterns may be reintroduced and the computational overhead of the inner loop remains fixed.

Several instances of bagging or boosting routines have appeared in GP, although the principle objective in each case was to improve model quality rather than minimize the cost of the inner loop. Specifically, a partitioned population model was utilized to construct ensembles of classifiers using both bagging and boosting by sampling [25]. This was then refined to produce boosting by weighting [26]. Both schemes were demonstrated under small benchmark applications. Naturally, the result is still an ensemble of weak classifiers and the inner loop is still performed over the entire dataset in each case.

An alternative approach to the problem formulates the task using constraint programming. The result is no longer a set of weak classifiers, but a single classifier developed by periodically reweighting the penalties in the cost function in proportion to the difficulty (error) of the current best individual of the population [27]. Such a scheme does not utilize an "age" penalty, as the inner loop retains the entire training dataset.

All the above cases emphasize a fixed training set with fixed size and single cost function. Host–parasite (or coevolutionary) models provide the potential for independent cost functions for both model (host) and dataset (parasite), where the (parasite) dataset, size P_s, is a subset of a larger training dataset P (sampled without replacement). The host models are now evolved over a smaller set of patterns than the entire dataset, where the content of such a subset varies over generations such that host (parasite) performance (difficulty) incrementally improves [28], [29].

As indicated in the introduction this work utilizes the concept of dynamic training subset selection [13], and then extends this into a hierarchy of subset selections. Efficient training over datasets in the order of hundreds of thousands of training patterns is now possible. In effect, the use of "blocks" ensures that the difficulty and age-based filtering of patterns only appears over a concise address range. Thus, temporal and spatial consistency in memory accesses is preserved—sampling a subset over the entire training dataset would encounter a significant overhead in terms of memory access time. There are, however, still several open design decisions. First, age and difficulty are only retained and developed over the duration of the block subset cycle. Each time a block changes the pattern age and difficulty are reset. Moreover, as the fitness of the population increases, the number of subset selections per block decreases (steps 2.c–2.e, Fig. 3), thus reducing the number of updates to pattern age and difficulty profiles. This means that as convergence approaches patterns for any block tend to be selected uni-

formly. Retaining age and difficulty would naturally imply that the pattern feature vector be extended by two across the entire training dataset—a solution that becomes increasingly undesirable as the dataset size increases. Future work will address this problem while maintaining independence from the dimension of the original dataset.

Naturally, one final issue of importance to any system that filters the training data based on pattern difficulty is the significance of outliers. The proposed system utilizes age as well as difficulty and resets the sampling biases for age and difficulty each time a block is replaced, thus providing a framework for reducing the significance of outliers. Such outliers may only be of significance as the number of outliers reaches a suitably significant proportion of the dataset. However, an open question remains as to the significance of outliers to difficulty-based sampling algorithms.

VIII. Conclusion

A framework for hierarchical DSS or RSS-DSS is detailed and successfully demonstrated on a training data set of 1/2 million patterns. The only critical parameter, which requires establishing, is the frequency with which blocks are refreshed (number of subsamples taken per block). The technique is independent of the data set and structure of GP employed. Moreover, the framework has no specialist hardware requirements, making use of the generic memory hierarchy design already widely supported in computing systems. Such a framework therefore has the potential to significantly improve the real-world application base of GP as a whole.

In the case of the application context—network borne anomaly detection—the significance of different cost functions is shown to favor the use of hierarchical or standard equally weighted fitness functions. Decoding of a specific solution is shown to provide both unique and intuitive decision rules with best-case detection rate on test data. In all cases, significant generalization is demonstrated, with attacks previously unseen detected, and attack classes with representation rates in training data of less than 1% also being detected. Moreover, around 10% of content-based attacks (U2R) appear to be detected in spite of not having direct access to explicit content-based features. To do so, it appears that the decision rules learnt are able to infer behaviors indirectly that are synonymous with content-based attacks. For example, the case of attempts to guess a password might be equivalent to multiple short duration connections between the same source and destination.

In terms of future work, the distributed detector scenario is of particular interest to intrusion detection. That is to say, the KDD-99 data set records connections across an entire network. Any detector trained on such a data set assumes global access to all information on the network. This is clearly not feasible in practice. Interesting possibilities therefore exist in terms of coevolutionary solutions in which multiple detectors cooperate to solve distributed detection problems (from a machine learning perspective such a scheme might correspond to the case of parallel bagged classifiers). With regards to the hierarchical RSS-DSS algorithm, two principle paths are of interest. First, at present the total number of block selections and therefore the stop criteria is currently fixed *a priori*. Establishing when all the block error rates converge, however, might well provide the basis for identifying an early stopping criterion. Second, "difficult" training patterns are currently limited to the lifetime of the corresponding block where block lifetimes decrease with improved classifier accuracy. Extending the current algorithm to provide a path by which difficult patterns may "live" longer than the current block would also be of interest.

TABLE VII
Summary of Top 16 Attacks for Test Set Performance of Individual With Best Case Detection Rate Under a Cost Function With Equal Error Weight

Previously Encountered Attacks		
Attack Type	% Misclassified	Total Patterns
Neptune	0	58,001
Portsweep	0	354
Land	0	9
Nmap	0	84
Smurf	0.08	164,091
Satan	3.55	1,633
Normal	3.27	60,577
Previously Unseen		
Udpstorm	0	2
Prosstable	3.0	759
Saint	5.98	736
Mscan	8.45	1,053
Httptunnel	15.82	158
Phf	50	2
Apache2	65.5	794

Appendix A

Given the relative simplicity of solutions identified by the equally weighted cost function (albeit at the expense of false positive rate), the GP solution with best-case detection rate was simplified and analyzed further. The decoded and simplified individual is detailed as follows:

$$\text{Connection}(t-24) = \frac{\frac{(20-\text{SRT}(t-16)\times\text{Protocol}(t-24)}{\text{DST}(t-24)} - \text{SRT}(t-16) - \text{SRT}(t)}{\text{DST}(t-24)}. \quad (3)$$

Table VII summarizes performance of the individual over a sample set of attack types from the test set in terms of attack types encountered during training (24 different types) and attack types only encountered during test (14 different types). We note classification accuracy is maintained for attack types, which are both previously encountered/not encountered, indicating that the solution does indeed display generalization to a wider context than that of the training set alone.

From (3), we conclude that the rule is estimating the statistics of the number of bytes from the responder and the byte ratio destination-source. This identifies that the attacking telnet connections in the DARPA dataset are statistically different from the normal telnet connections. Moreover, such a rule never misses an attack of "Neptune," "portsweep," "land," "nmap,", "udpstorm." It also provided "good performance on

"smurf," "processtable," "normal," "satan," "saint," "mscan," and "httptunnel." For "neptune," there are many half open TCP connections, without any data transfer. In "smurf," there are many echo replies to victim, but no echo requests from victim. In "http tunnel," the attacker defines attacks on the http protocol, which is normal, but the actual data exchange ratio makes it different from normal traffic. Currently, only [30] argued that telnet connection can be differentiated by a rule of the form discovered here. It has been suggested that attacks be formulated with such a rule in mind, [31], but without explicitly proposing using this statistic. Thus, GP in this case has actually provided a unique generic rule for the detection of multiple attack types. The relatively high FP rate of the rule (3.3%) would preclude the use of such a rule on its own, however, the simplicity of the solution would favor real-time applications, such as firewalls, where intrusion detection rules are beginning to appear as a first line of defense [32].

ACKNOWLEDGMENT

The authors gratefully acknowledge the efforts of the TRANSACTIONS Editor and anonymous reviewers for their constructive comments during the review of this paper, in particular regarding related works and the detailing of the hierarchical DSS-RSS algorithm itself. We also wish to extend our thanks to ThorSolutions, Inc. and the Telecommunication Research Application Research Alliance (TARA) for their in-kind support and encouragement during this research.

REFERENCES

[1] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zissman, "Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation," in *Proc. DARPA Inf. Survivability Confer. Exposition (DISCEX)*, vol. 2, 2000, pp. 12–26.

[2] J. McHugh, "Testing intrusion detection systems: A critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory," *ACM Trans. Inf. Syst. Security*, vol. 3, no. 4, pp. 262–294, 2000.

[3] C. Elkan, "Results of the KDD'99 classifier learning contest," *SIGKDD Explorations. ACM SIGKDD*, vol. 1, no. 2, pp. 63–64, 2000.

[4] L. Wenke, S. J. Stolfo, and K. W. Mok, "A data mining framework for building intrusion detection models," in *Proc. IEEE Symp. Security Privacy*, 1999, pp. 120–132.

[5] B. Pfahringer, "Winning the KDD99 classification cup: Bagged boosting," *SIGKDD Explorations*, vol. 1, no. 2, pp. 65–66, 2000.

[6] I. Levin, "KDD-99 classifier learning contest LLSoft's results overview," *SIGKDD Explorations*, vol. 1, no. 2, pp. 67–75, 2000.

[7] M. Vladimir, V. Alexei, and S. Ivan, "The MP13 approach to the KDD'99 classifier learning contest," *SIGKDD Explorations*, vol. 1, no. 2, pp. 76–77, 2000.

[8] T. Bass, "Intrusion detection systems and multisensor data fusion," *Commun. ACM*, vol. 43, no. 4, pp. 99–105, April 2000.

[9] F. H. Bennett III, J. R. Koza, J. Shipman, and O. Stiffelman, "Building a parallel computer system for $18 000 that performs a half peta-flop per day," in *Proc. Genetic Evol. Comput. Conf.*, 1999, pp. 1484–1490.

[10] H. Juillé and J. B. Pollack, "Massively parallel genetic programming," in *Advances in Genetic Programming: Volume 2*, P. J. Angeline and K. E. Kinnear, Eds. Cambridge, MA: MIT Press, 1996, ch. 17, pp. 339–358.

[11] J. R. Koza, F. H. Bennett, B. J. L. Bennett, S. L. Bade, M. A. Keane, and D. Andre, "Evolving computer programs using rapidly reconfigurable field programmable gate arrays and genetic programming," in *Proc. ACM 6th Int. Symp. Field Program. Gate Arrays*, 1998, pp. 209–219.

[12] P. Nordin, "A compiling genetic programming system that directly manipulates the machine code," in *Advances in Genetic Programming*, K. E. Kinnear, Ed. Cambridge, MA: MIT Press, 1994, ch. 14, pp. 311–334.

[13] C. Gathercole and P. Ross, "Dynamic training subset selection for supervised learning in genetic programming," in *Lecture Notes in Computer Science*. New York: Springer-Verlag, 1994, vol. 866, Parallel Problem Solving From Nature III, pp. 312–321.

[14] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 3rd ed. San Mateo, CA: Morgan Kaufmann, 2003. ISBN 1-55860-596-7.

[15] D. Song, M. I. Heywood, and A. N. Zincir-Heywood, "A linear genetic programming approach to intrusion detection," in *Lecture Notes in Computer Science*, vol. 2724, Proc. Genetic Evol. Comput. Conf., E. Cantú-Paz *et al.*, Eds., 2003, pp. 2325–2336.

[16] W. Lee and S. J. Stolfo, "A framework for constructing features and models for intrusion detection systems," *ACM Trans. Inf. Syst. Security*, vol. 3, no. 4, pp. 227–261, Nov. 2000.

[17] P. Lichodzijewski, A. N. Zincir-Heywood, and M. I. Heywood, "Host-based intrusion detection using self-organizing maps," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, 2002, pp. 1714–1719.

[18] L. Huelsbergen, "Finding general solutions to the parity problem by evolving machine-language representations," in *Proc. 3rd Conf. Genetic Programm.*, 1998, pp. 158–166.

[19] S. Luke and L. Panait, "Lexicographic parsimony pressure," in *Proc. Genetic Evol. Comput. Conf.*, W. B. Langdon *et al.*, Eds., 2002, pp. 829–836.

[20] R. M. Friedberg, "A learning machine: Part I," *IBM J. Res. Develop.*, vol. 2, no. 1, pp. 2–13, 1958.

[21] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs," in *Proc. Int. Conf. Genetic Algorithms and Their Applicat.*, 1985, pp. 183–187.

[22] M. I. Heywood and A. N. Zincir-Heywood, "Dynamic page-based linear genetic programming," *IEEE Trans. Syst., Man, Cybern. B. Cybern.*, vol. 32, no. 3, pp. 380–388, 2002.

[23] L. Brieman, "The heuristics of instability in model selection," *Annu. Statistics*, vol. 24, pp. 2350–2383, 1996.

[24] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, pp. 119–139, 1997.

[25] H. Iba, "Bagging, boosting and bloating in genetic programming," in *Proc. Genetic Evol. Comput. Conf.*, W. Banzhaf *et al.*, Eds., 1999, pp. 1053–1060.

[26] G. Paris, D. Robilliard, and C. Ronlupt, "Applying boosting techniques to genetic programming," in *Lecture Notes in Computer Science*, vol. 2310, Proc. 5th Int. Conf. Artif. Evol., P. Collet *et al.*, Eds., 2001, pp. 267–278.

[27] J. Eggermont, A. E. Eiben, and J. I. van Hemert, "Adapting the fitness function in GP for data mining," in *Lecture Notes in Computer Science*, vol. 1598, Proc. Eur. Workshop on Genetic Program., R. Poli, P. Nordin, and W. B. Langdon, Eds., 1999, pp. 193–202.

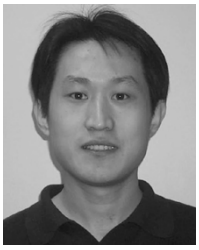
[28] W. D. Hillis, "Coevolving parasites improve simulation evolution as an optimization procedure," *Physica D*, vol. 42, pp. 228–234, 1990.

[29] J. Cartlidge and S. Bullock, "Learning lessons from the common cold: How reducing parasite virulence improves coevolutionary optimization," in *Proc. IEEE Congr. Evol. Comput. World Congr. Comput. Intell.*, vol. 2, 2002, pp. 1420–1425.

[30] J. B. D. Caberera, B. Ravichandran, and R. K. Mehra, "Statistical traffic modeling for network intrusion detection," in *Proc. 8th Int. Symp. Modeling, Anal., Simulation Comput., Telecommun. Syst.*, 2000, pp. 466–473.

[31] K. Kendall, "A database of computer attacks for the evaluation of intrusion detection systems," M.S. thesis, Massachusetts Inst. Technol., Cambridge, MA, 1998.

[32] Cisco IOS Firewall Intrusion Detection System Documentation (2004, Oct.). [Online]. Available: http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/120newft/120t/120t5/iosfw2/ios_ids.htm



Dong Song received the B.S. degree from Shanghai Jiao Tong University, Shanghai, China, in 1996 and the M.S. degree from Dalhousie University, Halifax, NS, Canada, in 2003, both in computer science.

He is currently a Senior Software Developer in the R&D Department, Quest Software, Inc., Halifax, NS, Canada. His research interests include evolutionary computation, linear genetic programming, pattern recognition, classification, data mining, and software engineering. In particular, he focuses on creating training systems scaling well on large data sets with small computational footprints, translating solutions into human readable format, and comparing them with human reasoning.

Malcolm I. Heywood (S'93–M'95) received the Ph.D. degree from the Department of Electronic Systems Engineering, University of Essex, Colchester, U.K., in 1994.

He is an Associate Professor of Computer Science, Dalhousie University, Halifax, NS, Canada. He has held faculty positions with the Department of Computer Science, Dokus Eylul University, Turkey (1998–2000), and with the IIMS Research Centre, School of Engineering, University of Sussex, Brighton, U.K. (1995–1998). He was a Research Fellow with the Neural Applications Group, Brunel University, London, U.K. (1994–1995). His principle research interests are in genetic programming, neural networks, and their application to real-world problems.

A. Nur Zincir-Heywood (S'94–M'99) received the Ph.D. degree in network information retrieval from the Department of Computer Engineering, Ege University, Izmir, Turkey, in 1998.

She is an Associate Professor with the Computer Science Department, Dalhousie University, Halifax, NS, Canada. From 1996 to 1997, she was a Visiting Researcher at the IIMS Research Center, School of Engineering, University of Sussex, Brighton, U.K. Previous to her current position, she was an Assistant Professor with the Department of Computer Engineering, Ege University (1998–2000). She has also been involved with Network Technology Workshops of Internet Society as an Instructor from 1997 and 2000. Her research interests include intrusion detection, network security, network management, and network information retrieval. She has published journal and conference papers in these areas, and has been involved in projects concerning network security and information systems.

Dr. Zincir-Heywood is a member of the Association for Computing Machinery (ACM).