A LINEAR GENETIC PROGRAMMING APPROACH TO INTRUSTION DETECTION

by

Dong Song

Submitted in partial fulfilment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
March 2003

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled " A Linear Genetic Programming approach to Intrusion Detection" by Dong Song in partial fulfilment of the requirements for the degree of Master of Computer Science.

Dated: _____

Supervisor: _____

Co-supervisor: _____

Readers: _____

_____

DALHOUSIE UNIVERSITY

DATE: March, 2003

AUTHOR: Dong Song

TITLE: A Linear Genetic Programming approach to Intrusion Detection

DEPARTMENT OR SCHOOL: Faculty of Computer Science

DEGREE: MSC CONVOCATION: May YEAR: 2003

Table of Contents

List of Figures

List of Tables

Abstract

Page-based Linear Genetic Programming (GP) is implemented with a new two-layer Subset Selection scheme to address the two-class intrusion detection classification problem on the KDD-99 benchmark dataset. By careful adjustment of the relationship between subset layers, over fitting by individuals to specific subsets is avoided. Unlike the current approaches to this benchmark, the learning algorithm is also responsible for deriving useful temporal features. Following evolution, decoding of a GP individual demonstrates that the solution is unique and comparative to hand coded solutions found by experts. Standard, dynamic and lexicographic fitness are implemented and compared.

In summary, this work represents a significant advance over previous applications of GP in which evaluation is limited to relatively concise benchmark datasets (typically less than a thousand patterns). This work details a hierarchical extension to the Subset Selection scheme resulting in the successful application of GP to a training dataset consisting of over 494,021 patterns. Moreover, the computational requirements are less than decision tree solutions applied to the same dataset.

Acknowledgements

I would like to gratefully acknowledge the ingenuity, patience, and helpful suggestion of Dr. Malcolm Heywood and Dr. Nur Zincir-Heywood.

Chapter 1: 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 intrusion detection systems (IDS) provides many challenges. In particular the environment is forever changing, both with respect to what constitutes normal behavior and abnormal behavior. Moreover, given the utilization levels of networked computing systems, it is also necessary for such systems to work with a very low false alarm rate [Lippmann, 2000].

In order to promote the comparison of advanced research in this area, the Lincoln Laboratory at MIT, under DARPA sponsorship, conducted the 1998 and 1999 evaluation of intrusion detection [Lippmann, 2000]. As such, it provides a basis for making comparisons of existing systems under a common set of circumstances and assumptions [McHugh, 2000]. Based on binary TCP dump data provided by DARPA evaluation, millions of connection statistics are collected and generated to form the training and test data in the Classifier Learning Contest organized in conjunction with the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining 1999 (KDD-99). The learning task 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 for the contest [Elkan, 2000] [Wenke, 1999]. 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 [Pfahringer,2000]. The second placed entry consisted of a decision forest containing 755 trees [Levin, 2000] The third placed entry consisted of two layers of voting decision trees augmented with human security expertise [Vladimir, 2000] There are other approaches, including data mining and statistical models.

In this work, the first interest is to explore a Genetic Programming (GP) approach to produce computer programs with a much less complex structure, compared with the above data mining approaches, yet yielding satisfactory performance on the KDD-99 test set. The motivation for such a goal is twofold. Concise rules are much easier to interpret. This provides for a greater level of confidence in the user base and facilitates the identification of weaknesses and strengths of the discovered rule(s). In addition simple rules provide the basis for higher throughput once implemented, where this has significant impact on real time applications such as intrusion detection. Genetic Programming has the potential to achieve this as solutions take the form of computer programs identified as part of a supervised learning framework with minimal *a priori* knowledge. Moreover, a lot of empirical evidence suggests that solutions found are often unique e.g. Chapter 57 in [Koza, 1999]. Second interest, the training system must scale well on a comparatively large training data set (i.e. there are approximately half a million patterns in the 10% KDD-99 training set). Given the computationally expensive inner loop of fitness evaluation in GP, this represents a challenging requirement in itself; Chapter 57, [Koza, 1999]. Third interest, fitness measurement must guide the training process towards to solutions which are able to generalize on patterns previously unseen i.e. we wish to explicitly avoid the case of a template based IDS system as these do not

scale well as the number of attack scenarios increases. Finally, only the most basic feature set is to be utilized. That is to say, the KDD data set provides a total of 41 connection features of increasing complexity. The utility of such a wide range of features makes the learning problem much easier, but has also been widely criticized [McHugh, 2000]. In particular most of the additional features are used to represent temporal properties. Thus, learning systems based on such a set of features are not required to learn any temporal characteristics to solve the problem (only structural credit assignment is necessary).

To this end, this work utilizes a Page-based Linear Genetic Program as the generic learning system. The training data 'sizing' problem is addressed through the use of Dynamic Subset Selection and Random Subset Selection [Gathercole, 1994] in combination with a new method for composing the problem as a hierarchy of subsets and preventing subset overlearning. The temporal feature-learning problem is addressed by letting GP index a 'window' of connection samples. Finally, generalization properties are investigated under three different fitness measures. In each case, best solutions are examined and compared with KDD-99 winning entries. Solutions developed from different fitness measurements are simplified by removing structural introns and analyzed. One solution is compared with rules extracted by domain experts on the same data set.

In the following text, Chapter 2 summarizes the properties associated with the KDD-99 IDS data set. Chapter 3 details the Genetic Programming (GP) approach taken, with a particular emphasis on the methodology used to address the size of the data set. Chapter 4 details parameter settings and conducts an evaluation of generalization under the case of

a standard fitness metric. This work will appear at GECCO 2003 [Song, 2003]. Chapter 5 details the methodology of dynamically weighting fitness for different attack classes and compares results with the standard metric. Chapter 6 details the methodology of lexicographic fitness measurement (hierarchical cost function) and compares all the results generated from different measurements. Finally, conclusions and future directions discussed in Chapter 7.

Chapter 2: Literature Survey and Background to Intrusion Detection

Section 2.1: Intrusion Detection Problem

As indicated in the introduction, from the perspective of the Genetic Programming (GP) paradigm, KDD-99 posted several challenges [Lippmann, 2000] [McHugh, 2000]. The amount of data is much larger than normally the case in GP applications. The entire training dataset consists of about 5,000,000 connection records. However, KDD-99 provided a concise training dataset – which is used in this work – and appears to be utilized in the case of the entries to the data-mining competition [Elkan, 2000] [Wenke, 1999] [Pfahringer, 2000] [Levin, 2000] [Vladimir, 2000]. Known as "10% training" this contains 494,021 records among which there are 97,278 normal connection records (i.e. 19.69 %). However, this still represents a considerable challenge for GP. Specifically, GP is a data driven learning algorithm (section 2.3) in which each candidate solution competes with others. To do so, each individual requires a scalar fitness, where this is proportional to error measured across the training data set. For example, if there are 4,000 individuals and 5- iterations of the algorithm, approximately 100×10^9 program evaluations will take place. Moreover, the stochastic nature of the algorithm requires that runs be conduced over at least 30 different initializations in order to establish the statistical significance of any results (i.e. verify that solutions are not due to random chance). Finally, we also note that reading the entire data set into computer cache is not feasible. That is to say, sequentially presenting each training pattern to a program for classification does not make use of the localized spatial and temporal access patterns on

which cache memory is based [Hennessy, Patterson 2002]. In this work we will therefore address these problems by making use of the observation that not all training patterns are equally difficult. Thus, a hierarchical competition between patterns is defined, based on the Dynamic Subset Selection technique [Gathercole, 1994].

As indicated in the introduction, the KDD'99 dataset describes each connection record in terms of 41 features and a label declaring the connection as either normal, or as a specific attack type. In this work, in order to naturally support the 2 address mode employed by the instruction format of L-GP [Heywood, 2002], of the 41 features, only the first eight (of nine) "Basic features of an Individual TCP connection"; hereafter referred to as 'basic features' are employed. The additional 32 *derived* features, fall into three categories,

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 second temporal window. One example of such a feature would be the number of connections to the same host over the 2 second 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 seconds.

In this work, none of these additional features are employed, as they appear to almost act as flags for specific attack behaviors. Our interest is on assessing how far the GP paradigm would go on 'basic features' alone.

The training data[1] encompasses 24 different attack types, grouped into one of four categories: User to Root; Remote to Local; Denial of Service; and Probe. Naturally, the distribution of these attacks varies significantly, in line with their function – 'Denial of Service,' for example, results in many more connections than 'Probe'. Table 2.1 summarizes the distribution of attack types across the training data[1]. Test data[2], on the other hand, follows a different distribution than in the training data, where this has previously been shown to be a significant factor in assessing generalization [Elkan, 2000]. Finally, the test data added an additional 14 attack types not included in the training data, as shown in Table 2.1, and therefore considered to be a good test of generalization [Elkan, 2000].

Table 2.1. 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%

Given that this work does not make use of the additional 32 derived features, it is necessary for the detector to derive any temporal properties associated with the current pattern, $x(t)$. To this end, as well as providing the detector with the labeled feature vector

1. "10% KDD dataset" in KDD contest [Lippmann, 2000].
2. "Corrected test set" in KDD contest [Lippmann, 2000].

for the current pattern, [$x(t)$, $d(t)$], the detector is also allowed to address the previous 'n' features at some sampling interval (modulo(n)).

Section 2.2: Machine Learning Approaches on KDD'99

In KDD-99 contest, there were 24 entries submitted, of which the three best approaches returned performance with essentially the same statistical significance [Elkan, 2000].

The "Winning entry" made use of an ensemble of 50x10 C5 decision trees, using cost-sensitive bagged boosting [Pfahringer, 2000]. In its training process, 50 samples were drawn from the original training set. The samples always included all of the examples of the two smallest classes 'U2R' and 'R2L', and 4000 PROBE, 80000 NORMAL, and 400000 DOS examples without duplicate entries. For each sample, an ensemble of 10 decision trees was built using both of the error-cost and boosting options. The final predictions were computed on top of the 50x10 trees with Bayes optimal prediction used to minimize conditional risk. Table 2.2 summarizes the result performance. Each sample took C5 a little less than an hour to process on 2 processors ultra-sparc2 (2x300Mhz) with 512M RAM, and a 9G disc running Solaris 5.6. Processing for all, yielding 50x10 trees, therefore took approximately 24 hours in the final "production" run.

Table 2.2. Performance of KDD-99 wining entry on KDD corrected test set [Elkan,2000]

Predicted Actual	Normal	Probe	DOS	U2R	R2L	%Correct
Normal	60262	243	78	4	6	99.5%
Probe	511	3471	184	0	0	83.3%
DOS	5299	1328	223226	0	0	97.1%
U2R	168	20	0	30	10	13.2%

R2L	14527	294	0	8	1360	8.4%
%Correct	74.6%	64.8%	99.9%	71.4%	98.8%	

The "Second placed" solution also used decision trees [Levin, 2000]. Training was conducted over the 10% training set and split into a series of "good" partitions, where "good" partition are partitions independent with each other [Levin, 2000]. A set of decision trees was constructed from these partitions. A problem specific global optimization criterion was then used to select the optimal subset of trees to give the final prediction. The global optimization criterion employed was designed to minimize the total cost of misclassifications, whilst taking into account parameters for reliability and stability of prediction in an attempt to minimize the overfitting problem. The hardware used consisted of one PC (Pentium II 350 MHz) with 128 MB of RAM. It took in total about 22 hours of machine time to complete the process of finding all the patterns. Performance is detailed in Table 2.3.

Table 2.3. Performance of KDD-99 second entry on KDD corrected test set [Levin,2000]

Predicted Actual	Normal	Probe	DOS	U2R	R2L	%Correct
Normal	60244	239	85	9	16	99.42%
Probe	458	3521	187	0	0	84.52%
DOS	5595	227	224029	2	0	97.47%
U2R	177	18	4	27	2	11.84%
R2L	14994	4	0	6	1185	7.32%
%Correct	73.95%	87.83%	99.88%	61.3%	98.5%	

The "Third placed" solution was two layer decision trees [Vladimir, 2000]. The first layer was trained on the connections which can not be classified by security experts, whereas the second layer was built on the connections which can not be classified by first layer. Training data set was again the "10% KDD data set" and some of the DOS and

"normal" connections were randomly removed. This training dataset was then randomly split into three sub samples: 25% for tree generation, 25% for tree tuning, and 50% for estimating model quality. Two computers were used. One was a Pentium Notebook 150 MHz with 32 MB RAM and about 200MB free disk space, and another is Pentium Desktop 133 MHz with 40 MB RAM and about 200MB free disk space. It took about 30 minute to generate one set of trees. The total time for all calculations was about 6 hours.

There are other successful approaches on the classification problem of KDD-99. In [Wenke, 1999], a data-mining framework is described in which four data mining programs are used together to give a final classification as follows.

1. A Classification system, RIPPER, is used to classify connections into one of a 5 categories. RIPPER searches for the most discriminating features out of 42 features in KDD dataset and uses them to generate rules.
2. Meta-learning is used as a mechanism to inductively learn the correlation of predictions by a number of classifiers. The resulting metaclassifier thus combines the detection power of all the classifiers.
3. Association rules. The goal of mining association rules is to derive correlations between 41 connection features.
4. Frequent Episodes are used to represent the sequential audit record patterns.

The above framework performs as well as the best systems built using the manual knowledge engineering approaches [Wenke, 1999]. Table 2.4 compares the detection rates of old intrusions and new intrusions where new intrusions refer to those that did not appear in the training data. Table 2.5 summarizes the complexity of classifiers in terms of the number of features in a connection record, the number of

RIPPER rules produced, and the number of distinct features actually used in the rules. The numbers in bold, for example, 9, indicate the number of automatically constructed temporal and statistical features being used in the RIPPER rules.

Table2.4. Detection Rates [Wenke, 1999]

Category	%Old	%New
DOS	79.9	24.3
Probe	97.0	96.7
U2R	75.0	81.8
R2L	60.0	5.9
Overall	80.2	37.7

Table 2.5. Complexity of Classifiers [Wenke, 1999]

Model	# of features in records	# of rules	# of features in rules
content	22	55	11
traffic	20	26	4+**9**
Host trafic	14	8	1+**5**

In [Caberera, 2000], statistics of the traffic are modeled and used to recognize connection types in the KDD-99 dataset. Two statistics is used. The first is the number of bytes from the responder and the other is the byte ratio between responder and originator. A Kolmogorov-Smirnov test is then used to show that attacks using telnet connections in the DARPA dataset form a population that is statistically different from the normal telnet connections. The FP and detection rates are also comparable to other approaches obtained in the DARPA evaluation.

Section 2.3: Introduction to Genetic Programming

Evolutionary Computation mimics aspects of Darwinism's natural selection and survival of fittest to optimize a population of candidate solutions towards a predefined

fitness. During the evolutionary process, the combination of selection and search operators are used to generate fitter solutions. Genetic Programming (GP) is one instance of evolutionary computation in which an individual takes the form of executable code, hence "running" the program determines an individuals' fitness. In order to apply GP, it is necessary to define the 'instructions' from which programs are composed – often referred to as the Functional Set. The principle constraint on such a set being that it should provide syntactic closure and require one or more arguments. In addition, a Terminal Set is provided consisting of zero argument instructions, typically representing inputs from the environment or constants. The content of a population is modified through the application of selection and search operators. Selection operators govern how individuals from the population are sampled, thus giving rise to a competition between parents for 'space' in the population (population size is fixed). Search operators utilize the concepts of crossover and mutation from generation to introduce variation in children. Holland placed such a framework in a theoretical context and demonstrated for the case of individual composed from binary strings (Genetic Algorithm) [Holland, 1975]. The traits from the best performing individual will see expanded rates of reproduction, leading to monotony improvement to a population from then. This is the GA Schema Theorem; recent work by Poli has begun to provide equivalent relations for GP [Poli, 2001]. The method as a whole is popularized by the seminal work of Koza [Koza, 1989], [Koza, 1992] in the form of tree-structured GP. However, linearly structured GP has also been proposed [Freidburg, 1958] [Cramer, 1985], [Nordin, 1994], [Huelsbergen, 1996] [Heywood, 2002]. Familiarity with the linear structure resulted in the use of it in this work.

Section 2.3.1: Tree and Linearly Structured Genetic Programming

As indicated above, a linearly structured GP, or L-GP, takes the form of a 'linear' list of instructions. Execution of an individual mimics the process of program execution normally associated with a simple register machine as opposed to traversing a tree as in the case of tree-structured GP (leaves representing an input, the root node the output). Each instruction is constructed by an opcode and operand, and modifies the contents of internal registers, memory and program counter. Another aspect which differentiates T-GP and L-GP is in the operation of crossover. As shown in Fig 2.1 and Fig 2.2, T-GP individuals exchange sub trees between parents, while L-GP individuals exchange 'linear' sequences of instructions.

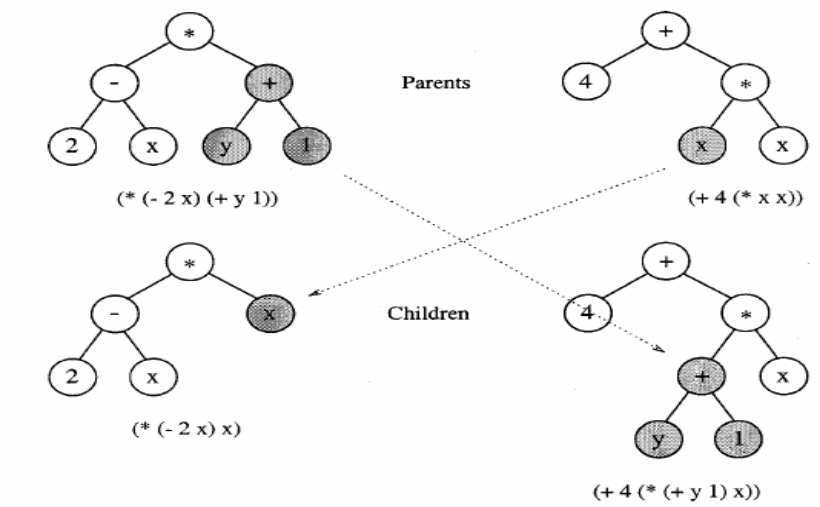

Fig 2.1. Crossover in Tree GP [Brameier, 2001]

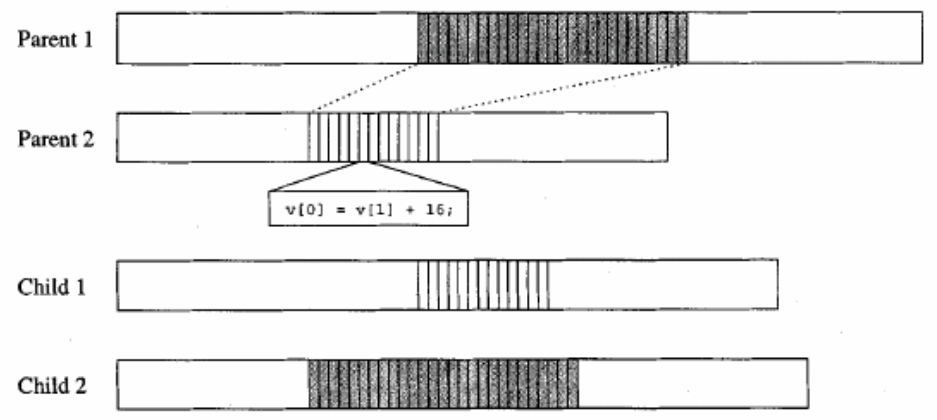

Fig 2.2. Classical crossover in Linear GP result in variable length individuals [Brameier,2001]

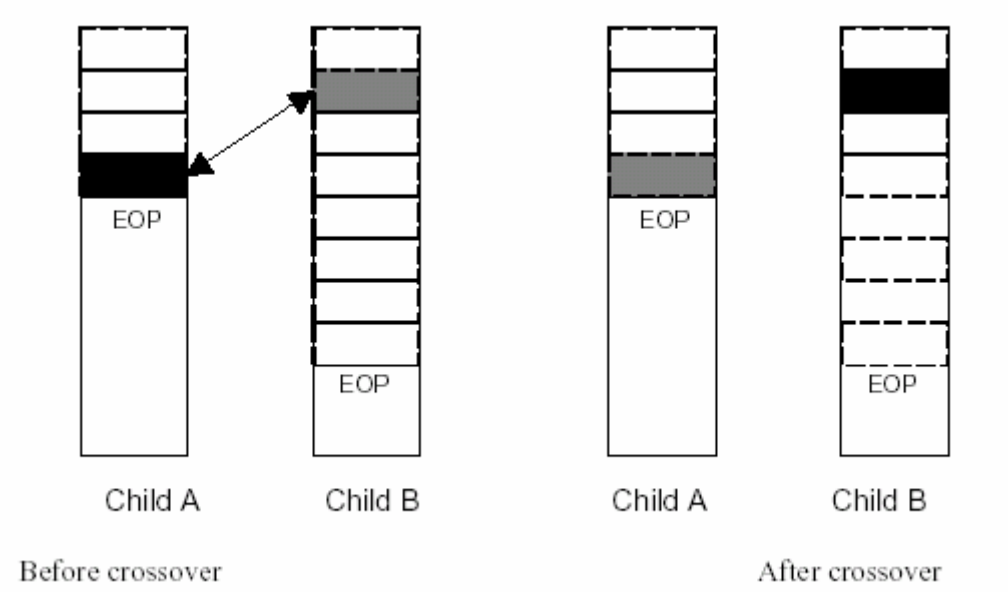

Fig2.3. Crossover in Page-based L-GP result in fixed length individuals [Heywood,2002]

Section 2.3.2: Page-based Linear GP with Dynamic Crossover

The operator crossover as typically applied in GP is blind. That is to say the stochastic nature of defining a crossover point results in individuals whose instruction count continues to increase with each generation without a corresponding improvement in performance. Page-based L-GP provides a method for encouraging the identification of

concise solutions without penalizing generalization ability [Heywood, 2002] [Wilson, 2002]. To do so, Page-based L-GP defines individuals in terms of a number of program pages selected stochastically at initialization, but thereafter does not change and a page size, as measured in terms of instructions per page (the same for all members of the population). The crossover operator merely selects which pages are swapped between two parents, where it is only possible to swap single pages. This means that following the initial definition of the population; the length of an individual never changes (length measured in terms of the number of pages and instructions per page). The number of pages each individual may contain is selected at initialization using a uniform distribution over the interval [1, max program length]. This is different from classical L-GP as: (1) the concept of pages does not exist; and (2) the number of instructions crossed over in classical L-GP is not constrained to be equal, resulting in changes to the number of instructions per individual. Moreover, any two parents denote crossover boundaries, thus alignment of code between boundaries never takes place [Huelsbergen, 1996], [Nordin, 1994].

Given that the page-based approach fixes the number of instructions per page, it would be useful if manipulation of the number of instructions per page was possible without changing the overall number of instructions per individual. The selection of different page sizes is related to the overall fitness of the population. It starts with smallest page size to encourage the identification of building blocks of small code sequences. When the fitness of the population reaches a ‘plateau,’ the page size is increased. Further plateaus make page size increased towards a maximum number. After then such cycles restart at the smallest page size. Such a scheme was denoted dynamic

page-based L-GP and extensively benchmarked against classical L-GP and tree structured GP in [Heywood, 2002]. In all cases concise solutions were located without penalizing generalization.

Section 2.3.3: Mutation Operation in L-GP

L-GP as employed here utilized two forms of mutation. The first type of mutation is used to manipulate the contents in individual instructions. To do so, an instruction is randomly selected, and then, an X-OR operation performed with a second randomly generated integer to create the new instruction, as shown in Fig 2.4. The principle motivation for such an operator is to introduce additional (code) diversity into the population.

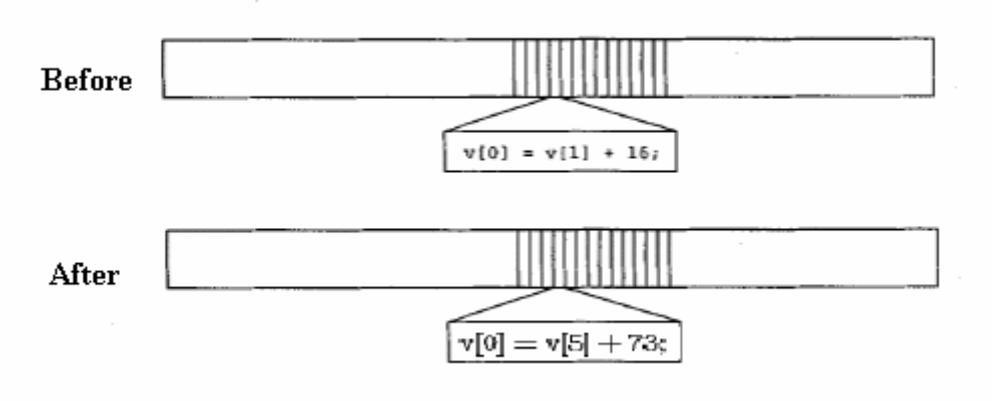

Fig 2.4. Instruction wild mutation

The second type of mutation operator was introduced to enable variation in the order of instructions in an individual. This is referred to as a swap operator. In this case, an arbitrary pair wise swap is performed between two instructions in the same individual, as shown in Fig 2.5. The motivation here is that the sequence, has a significant effect on the solution as well as the composition of instructions. Both types of mutation occur by probability threshold defined a priori and held constant throughout.

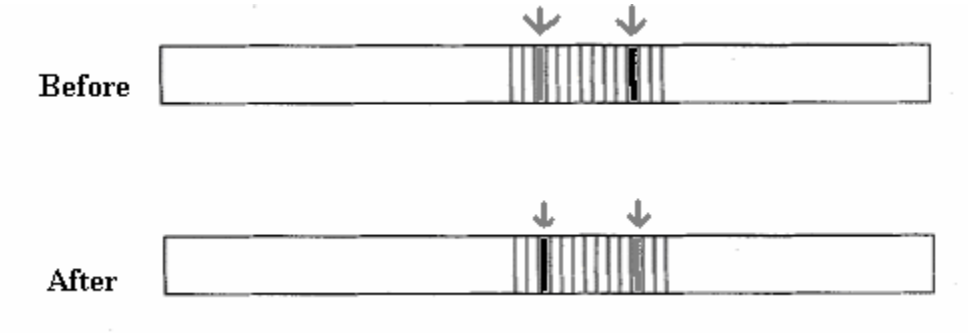

Fig 2.5. Swap operation

Section 2.3.4: Steady State Tournament Selection

Classically, both genetic algorithm and programming utilized a proportional selection operator [Holland 1975], [Koza 1989, 1992]. As such, at each generation M/2 pairs of parents are selected, where the probability of selecting individual *i,* P_i is

$$P_i = \frac{f_i(t)}{\sum_j f_i(t)} \qquad (1)$$

Thus at each generation *M* individuals are evaluated through reproduction, crossover and/or mutation. Concisely, steady state selection stochastically selects *K* (<<*M*) individuals independent of their current relative fitness. Each individual within the tournament is evaluated. The relative fitness of the *K* individuals is used to divide the tournament into two groups. The better half will be parents and remain in the population. The inferior half will be replaced by offspring created by stochastically applying the search operators to the parents. Offspring replace the *K* / 2 inferior members from the population. Both selection schemes have been extensively analyzed with Steady State tournament demonstrate a higher selection pressure [Syswerda, 1991]

Section 2.3.5: Functional Set

Functional and terminal sets in linear GP take the form of opcodes in the register-level transfer language instructions. Table 2.6, illustrates the difference between 0, 1, 2, 3 address formats in the case of a solution to the 'simple regression' benchmark problem. Note the opcodes, or function set, in this example is {+, -, x, %, NOP, EOP}, where NOP denotes no operation and EOP denotes end of program.

Table 2.6. An example of L-GP [Heywood, 2002]

Evaluating $x^4 + x^3 + x^2 + x$ using different register addressing modes.

0-address	1-address	2-address	3-address
TOS ← P0	AC ← P0	R1 ← P0	R1 ← P0 * P0
TOS ← P0	AC ← AC * P0	R1 ← R1 * R1	R1 ← R1 + P0
TOS ← TOS_t * TOS_{t-1}	AC ← AC + P0	R1 ← R1 + P0	R2 ← R1 * P0
R1 ← TOS	AC ← AC * P0	R2 ← R1	R2 ← R1 * P0
TOS ← R1	AC ← AC + P0	R2 ← R2 * P0	
TOS ← P0	AC ← AC * P0	R2 ← R2 * P0	
TOS ← TOS_t * TOS_{t-1}	AC ← AC + P0	R1 ← R1 + R2	
TOS ← P0			
TOS ← TOS_t + TOS_{t-1}			
TOS ← R1			
TOS ← R1			
TOS ← TOS_t * TOS_{t-1}			
TOS ← TOS_t + TOS_{t-1}			
TOS ← R1			
TOS ← TOS_t + TOS_{t-1}			

KEY: TOS – top of stack; R× - internal register ×; P× - external input on port ×; AC – accumulator register; {*, +}- arithmetic operators; ← - assignment operator.

Section 2.3.6: Terminal Set

The terminal set takes form of operands (e.g. permitted inputs, outputs and internal registers or general address space) in the register-level instructions. In addition, constants may also be expressed directly as a 'b' bit integer over the range 0 to $2^b - 1$, or $-2^b/2$ to $2^b/2 - 1$. In the above example, terminal set is {R1, R2, P0 (= x)}.

Section 2.3.7: Raw Fitness

Raw fitness represents the initial measurement of error as calculated between desired value (supplied by the data set) and value suggested by GP. As such raw fitness provides the basis for evaluating performance of a program. In the above example, the raw fitness was the sum of the absolute differences between the desired value and the actual result achieved from running the program. Thus, lower raw fitness means better performance. Regarding on the problem, a wrapper might be necessary to interpret the GP output in terms of the data. In particular, classification problem type requires the GP response to be expressed in terms of greater than zero (class 1) or less than zero (class 2). A count of the number of correct classifications is then facilitated.

Section 2.3.8: Training Termination Criteria

The training process terminates if either convergence is found, or if the number of generations exceeds a pre-set limit. Convergence is typically expressed in terms of minimizing (maximizing) a suitable cost function, where this is naturally a function of the application domain. Thus, an individual with a perfect result (say a raw fitness of zero), or performance within some predefined threshold of an ideal result. Typical cost functions might include a count for the number of correctly classified patterns or cases with error smaller than some predefined threshold. Moreover, the cost function may also include additional properties, such as a term for solution complexity. In this work several cost functions are investigated, sections 4, 5 and 6.

Section 2.3.9 Liner GP algorithm

Figure 2.6 summarizes the overall algorithm for page-based L-GP using dynamic crossover using pseudo codes. The training parameters used to initialize the training system consist of predefined thresholds for the probability of mutation, crossover and swap as well as the maximum number of pages in an individual and the maximum number of instructions per page.

```
Training(Parameters)
{     …
      initialize training system and population;
      read dataset into memory;
   …
   iteration = 0;
      while ( termination(iteration) == False)
      {
            conduct tournament selection;
            train selected individuals on the dataset;
            evaluate fitness;
            test for change in cross over page size;
            if (change == true) update page size;
            let offspring = parents;
            crossover between offspring;
            mutation on offspring;
            swap on offspring;
```

```
          overwrite worst members of tournament with
offspring;
          update population;
          iteration++;
     }
…}
```

Fig 2.6. Page-based L-GP with dynamic crossover

The initial population is generated randomly from the functional set and terminal set by first stochastically selecting instruction type (source reference is an internal register; source reference is a data value; specification of a constant) and then stochastically selects from the set of operands and opcodes to form a legal instruction. Note, the ratio of instruction types is also defined *a priori*.

Chapter 3: Methodology

The specific interest of this work lies in identifying a solution to the problem of efficiently training with a large dataset (close to half a million patterns). To this end, we revisit the method Dynamic Subset Selection [Gathercole, 1994]. Dynamic Subset Selection (DSS) involves randomly selecting a target number of cases from the whole training set every generation, with a bias towards patterns which have a history of being either 'difficult' to classify or has not been selected 'recently'. Random Subset Selection involves randomly selecting a target number of cases with equal probability.

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 (section 2.1); and the overhead associated with datasets that do not fit within RAM alone (section 2.1). In this work, as shown in Fig 3.1, a hierarchy is defined in which the data set is first partitioned into blocks. The blocks are small enough for retention in RAM where the result of a competition is a sub set of training patterns that reside within cache alone, and then a competition is initiated between training patterns within a selected block. The selection of blocks is performed using Random Subset Selection (RSS) – layer 1 in the hierarchy. Dynamic Subset Selection (DSS) enforces a competition between different patterns – layer 2 in the hierarchy. Thus, a hierarchical architecture has been defined in conjunction with the previous concepts of random and dynamic sub-set selection in order to facilitate access to a much larger training set (RSS and DSS were previously only demonstrated on a dataset of hundreds of patterns, therefore never requiring a hierarchy [Gathercole, 1994].)

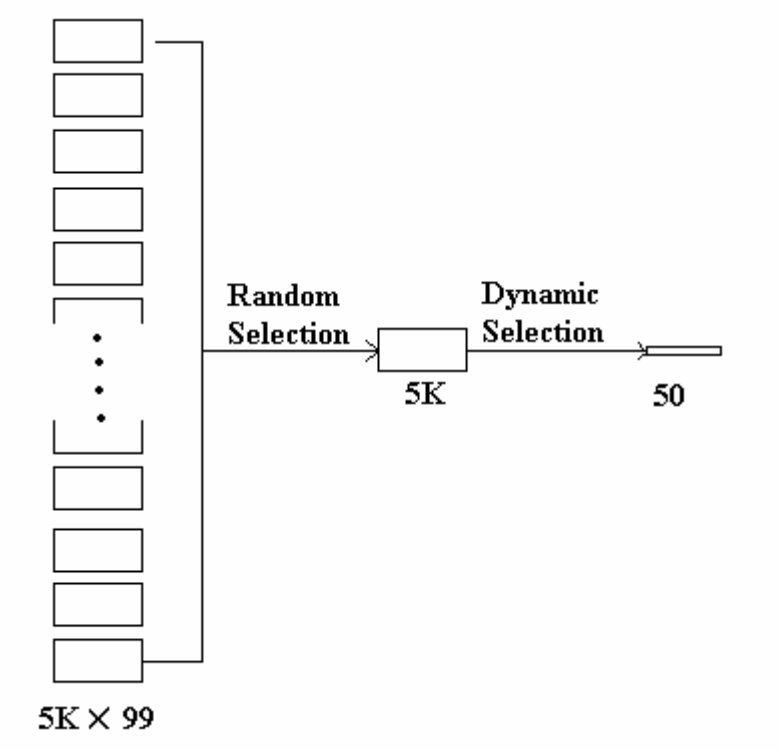

Fig 3.1. Two Layers of Subset Selection

Section 3.1: Subset Selection

First layer The KDD-99 10% training data set was divided into 99 blocks with 5000 connection records per block[2]. The size of such blocks is defined to ensure that, when selected, they fit within the available RAM. Blocks are randomly selected with uniform probability. Once selected, the history of training pressure on a block is used to define the number of iterations performed at the layer 2 in DSS. Such a training pressure is defined in proportion to the performance of the best-case individual. Thus, iterations of DSS, I, in block, b, at the current instance, i, is

$$I_b(i) = I_{(\max)} \times E_b(i-1) \qquad (5)$$

[2] Last block has 3993 connections.

where $I_{(max)}$ is the maximum number of subsets selected on a block; and $E_b(i-1)$ is the number of misclassifications of the best individual on the previous instance, i, of block, b. Hence, $E_b(i) = 1 - [\text{hits}_b(i) / \#\text{connections}(b)]$, where $\text{hits}_b(i)$ is the hit count over block 'b' for the best case individual identified over the last DSS tournament at iteration 'i' of block 'b'; and $\#\text{connections}(b)$ is the total number of connections in block 'b'.

Second layer A simplified DSS is deployed in this layer. That is, fixed probabilities are used to control the weighting of selection between age and difficulty. For each record in the DSS subset, there is a 30% (70%) probability of selecting on the basis of age (difficulty). Thus, a greater emphasis is always given to examples that resist classification. DSS utilizes a subset size of 50, with the objective of reducing the computational overhead in performing a particular fitness evaluation. Moreover, in order to further reduce computation, the performance of parent individuals on a specific subset is retained. Finally, after 6 tournaments the DSS subset will be reselected.

DSS Selection In the RSS block, every pattern is associated with an age value, which is the number of DSS selections since last selection, and a difficulty value. The difficulty value is the number of individuals that were unable to recognize a connection correctly the last time that the connection appeared in the DSS subset. Connections appear in a specific DSS stochastically, where there is 30% (70%) probability to select by age (difficulty). Roulette wheel selection is then conducted on the whole RSS block, with respect to age (difficulty). After the DSS subset is filled, age and difficulty of selected connections are reset to an initial value. For the rest, age is increased by 1 and difficulty remains unchanged.

Algorithm The following pseudo code describes the L-GP system with Dynamic Subset Selection. The lines in bold are additional code to support the hierarchy of subset selection where as the remainder defined in section 2.3.9

```
Training(Parameters)
{
     …
     initialize training system and population;
   RSSiteration = 0;
   while ( RSStermination(RSSiteration) == False)
     {
          randomly load one block b into memory;
      …
      iteration = 0;
     while ( termination(iteration) == False)
     {
          conduct tournament selection;
          conduct Dynamic Subset Selection;
          train individuals on DSS subset;
               test for change in cross over page size;
               if (change == true) update page size;
          let offspring = parents;
          crossover between offspring;
          mutation on offspring;
          swap on offspring;
```

```
            overwrite worst members of tournament with
    offspring;
            iteration++;
          }
           test the best individual against RSS block b;
           record number of errors on block b;
           Calculate max iterations of b;

    }
       …
    pick the best individual;
    remove introns;
    test it against whole dataset;
    record result and translate;
}
```

Section 3.2: Parameterization of the Subsets

The low number of patterns actually seen by a GP individual during fitness evaluation, relative to the number of patterns in the training data set, may naturally lead to 'over fitting' on specific subsets. Our general objective was therefore to ensure that the performance across subsets evolved as uniformly as possible. The principle interest is to identify the stop criterion necessary to avoid individuals that are sensitive to the composition of a specific Second Level subset. Note however that there is a natural conflict between reducing over fitting (equivalent to maximum diversity in patterns

utilized for fitness evaluation) and minimizing computational overhead in matching the evaluation (GP inner loop).

To this end, an experiment is conducted in which 2,000 block selections are made with uniform probability. In the case of each block selection, there are 400 DSS selections. Before selection of the next block takes place, the best performing individual (with respect to sub-set classification error) is evaluated over *all* patterns within the block, let this be the block error at selection i, or $E_b(i)$. In Fig 3.2, block errors over the first 500 selections are plotted and a linear least-squares regression is performed. The regression line indicates that the general trend in the first 500 selections is towards a block error converging tending to zero.

To further understand how performance changes at each selection, a sliding window is then constructed consisting of 100 block selection errors, and a linear least-squares regression performed. The *gradient* of each linear regression is then plotted, Figure 3.3 (1900 points). A negative trend implies that the block errors are decreasing whereas a positive trend implies that the block errors are increasing (the continuous line indicates the trend). It is now apparent, that after the first 750 block selections, the trend in block error has stopped decreasing. After 750 selections, oscillation in the block gradients appears, where this becomes very erratic in the last 500 block selections.

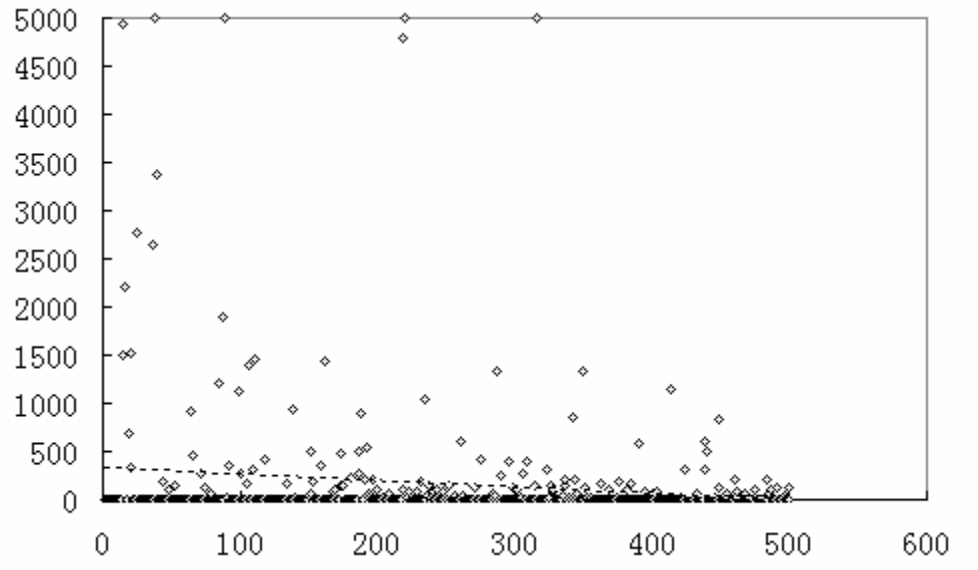

Fig 3.2 Block errors on first 500 iterations. X-axis represents tournament and Y represents block error

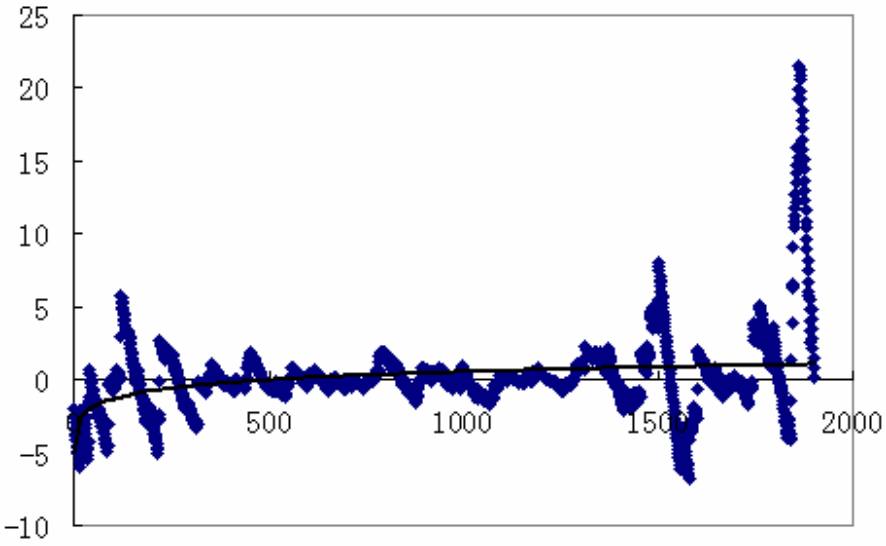

Fig 3.3. Gradient of block error using best case DSS individual. X-axis represents tournament and Y represents slope of best fitting line on 100 point window

On the basis of these observations, the number of DSS selections per block is limited to 100 (from 400) – with the objective of further reducing any tendency to prematurely

specialize – where the principle cost is in a lower number of DSS per block, thus increasing the number of times that the cache is flushed.

Section 3.3: Structural Removal of Introns

Introns are instructions, which have no influence on the output, but appear to be a factor in the evolution of solutions. Moreover, two forms of introns are often distinguished: structural introns and semantic introns. Structural introns manipulate variables that are not used for the calculation of the outputs at that program position. Whereas, semantic introns manipulate variables on which the state of the program is invariant [Brameier, 2001]. In this work, structural introns are detected using following pseudo code, initiated once evolution is complete, with the last reference to R0 (the output) as the input argument.

```
markExon(reg, i)
{
      ...;
   for ( ; destination in the i^th instruction != reg ; i--)
      if (i = 0 ) exit;
      mark i^th instruction as exon
      markExon(oprand1, i-1)
      markExon(oprand2, i-1)
      ....
}
```

Chapter 4: Evolving Individuals with Standard Fitness

In this initial experiment, equal penalty is assumed between errors made by GP individuals no matter the connection type. The following experiments are based on 40 runs using Dynamic Page-based L-GP. Runs differ only in their choice of a random seed initializing the initial population. That is, the same genetic page-based L-GP parameters are employed as in previous study [Heywood, 2002]. Functional set is selected with a bias towards simplicity. Table 4.1 lists the common parameter settings for all runs. The total number of records in training and test sets are summarized listed in Table 4.2. The method used for encouraging the identification of temporal relationships and composing the instruction set is defined as follows, and is reflected in the selection of Terminal set, Table 4.3, Table 4.4.

Table 4.1. Parameter Settings for Dynamic Page based Linear GP

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
Instruction type 2 probability	4
Instruction type 3 probability	1
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
Cost function	Increment by 1 for each

misclassification

Table 4.2. Distribution of Normal and Attacks

Connection	Training	Test
Normal	97249	60577
Attacks	396744	250424

Section 4.1: Implementation

Sequencing Information As indicated in section 2.1, only the 8 basic features of each connection are used, corresponding to: 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); LAND (1 if connection is from/to the same host/port, 0 otherwise); and number of "wrong" fragments (WRONG) as shown in Table 4.3. This implies that GP is required to determine the temporal features of interest itself. To do so, for each 'current' connection record, $x(t)$, GP is permitted to index the previous 32 connection records relative to the current sample t, modulo 4. Thus, for each of the eight basic TCP/IP features available in the KDD-99 dataset, GP may index the 8 connection records $[(t), (t - 4), \ldots (t - 28)]$, where the objective is to provide the label associated with sample 't' as show in Table 4.4.

Table 4.3 8 Basic features

Dur	Protocol	Service	Flag	DST	SRC	LAND	WRONG	**t**
…	…	…	…	…	…	…	…	
Dur	Protocol	Service	Flag	DST	SRC	LAND	WRONG	**t - 4**
…	…	…	…	…	…	…	…	
Dur	Protocol	Service	Flag	DST	SRC	LAND	WRONG	**t - 8**
…	…	…	…	…	…	…	…	
…	…	…	…	…	…	…	…	
…	…	…	…	…	…	…	…	
Dur	Protocol	Service	Flag	DST	SRC	LAND	WRONG	**t - 32**

Table 4.4 Temporal features

T	**t - 4**	**t – 8**	**t – 12**	**t -16**	**t - 20**	**t - 24**	**t - 28**

Instruction Set A 2-address format is employed (Table 2.6) in which provision is made for: up to 16 internal registers, up to 64 inputs (Terminal Set), 5 opcodes (Functional Set) – the fifth is retained for a reserved word denoting end of program – and an 8-bit integer field representing constants (0-255), as shown in Table 4.5. Two mode bits toggle between one of three instruction types: opcode with internal register reference; opcode with reference to input; target register with integer constant. Extension to include further inputs or internal registers merely increases the size of the associated instruction field. The output is taken from the first internal register.

Table 4.5. Instruction format

Operation Code	Destination Register (0 – 7)	Source Register (0- 7) | Terminal [row][column] | const (0 – 255)
3 bits	3 bits	6 bits

Section 4.2: Results and Analysis

Training was performed on a Pentium III 1GHz platform with a 256M byte RAM under Windows 2000. It takes approximately 15 minutes to finish each trial. The 40 best individuals within the last tournament are recorded and simplified as per Section 3.3. Performance is expressed in terms of average program length (before and after simplification), false positive (FP) and detection rates, estimated as follows,

$$Detection\,Rate = 1 - \frac{\#False\ Negatives}{Total\ Number\ of\ Attacks} \quad (3)$$

$$False\ Positve\ Rate = \frac{\#False\ Positives}{Total\ Number\ of\ Normal\ Connections} \quad (4)$$

Figure 4.1 summarizes the performance of all 40 runs in terms of FP and Detection rate on both training and test data. Of the forty cases, three represented outliners (not plotted). That is to say, they basically classified everything as normal or attack. Outside of three outliner cases, it is apparent that solution classification accuracy is consistently achieved. Table 4.6 makes a direct comparison between KDD-99 competition winners, verses the corresponding GP cases.

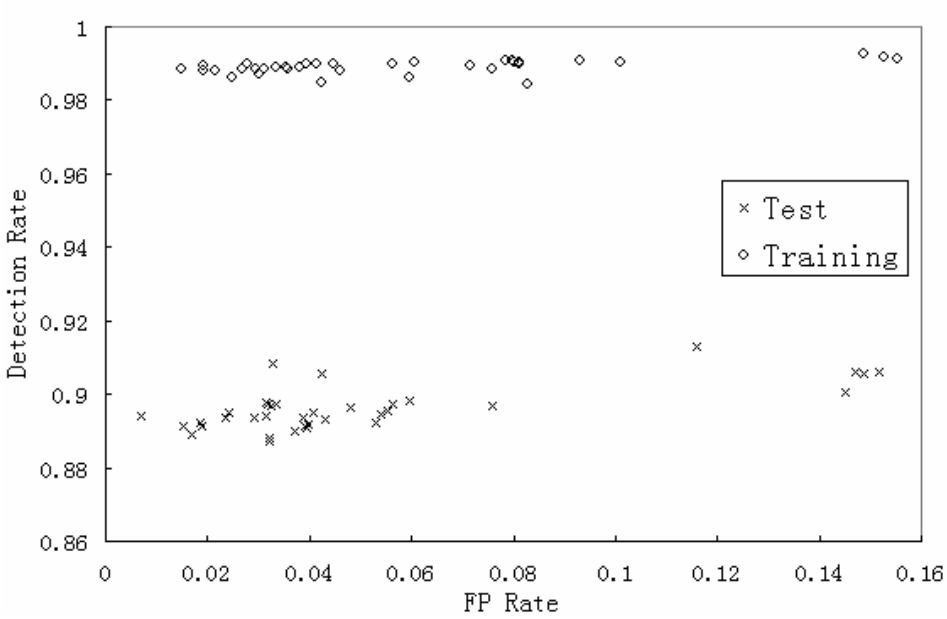

Fig 4.1. FP and detection rate of 40 runs on KDD-99 test data set

Table 4.6. Comparison with KDD-99 winning entries

Parameter	Detection Rate	FP rate
Winning entry	0.908819	0.004472
Second place	0.915252	0.00576
Best FP rate GP	0.894096	0.006818
Best Detection rate GP	0.908252	0.032669

Structural removal of introns, Section 3.3, resulted in a 5:1 decrease in the average number of instructions per individual (87 to 17 instructions). With the objective of identifying what type of rules were learnt, the GP individual with best Detection Rate from Table 4.6 was selected for analysis. Table 4.7 lists the individual following removal

of the structural introns. Table 4.8 summarizes performance of the individual over a sample set of the connection types in terms of connections types seen during training (24 different types) and connections types only during test (14 different types). Of particular interest here is that high classification accuracy is returned for connection types, which are both frequent and rare, where it might be assumed that only the connections with many examples might be learnt.

Table 4.7. Anatomy of Best Individual

Opcode	Destination	Source
LOD	R[0]	20
SUB	R[0]	Input[2][5]
MUL	R[0]	Input[0][1]
DIV	R[0]	Input[0][4]
SUB	R[0]	Input[2][5]
SUB	R[0]	Input[6][5]
DIV	R[0]	Input[0][4]

Table4.8 Error rates on test data for top 16 attacks by individual with Best Detection Rate

Seen connection type	% Misclassified	Total Examples	Unseen connection type	% Misclassified	Total Examples
Neptune	0	58,001	Udpstorm	0	2
Portsweep	0	354	Prosstable	3.03	759
Land	0	9	Saint	5.978	736

Nmap	0	84	Mscan	8.452	1,053
Smurf	0.077	164,091	Httptunnel	15.823	158
Satan	3.552	1,633	Phf	50	2
Normal	3.267	60,577	Apache2	65.491	794

Re-expressing this individual analytically provides the expression,

$$Output = \frac{\frac{(20 - Input[2][5]) \times Input[0][1]}{Input[0][4]} - Input[2][5] - Input[6][5]}{Input[0][4]}$$

It is now apparent that the statistics for the number of bytes from the destination and the byte ratio destination-source are utilized. This enables the individual to identify that the attacking telnet connections in the DARPA dataset are statistically different from the normal telnet connections. Moreover, not only telnet connections can be classified by this way. Such a rule never misses an attack of "Neptune"(DOS), "portsweep"(Probe), "land"(DOS), "nmap" (Probe), "udpstorm"(DOS). It also had good performance on "smurf"(DOS), "processtable" (DOS), "normal"(Normal), "satan"(Probe), "saint"(Probe), "mscan" (Probe) and "httptunnel" (R2L). Moreover, 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 [Caberera, 2000] *argued* that telnet connections might be differentiated by a rule of the form discovered here. Moreover, it has also been suggested that attacks be formulated with such a rule in mind, [Kendall, 1998], but without explicitly proposing this statistic. Thus GP in this case has actually

provided a unique generic rule for the detection of multiple attack types before such a rule was openly formulated.

Chapter 5: Dynamic Fitness – Motivation and Performance

In the previous experiment, penalties for errors across all the connection types were weighted equally. However, this does not actually reflect the distribution of connection types in the training set. Thus, GP individuals which had better performance on frequently occurring connection types will be more likely to survive, even if they perform worse than competing individuals on the less frequent types. From Table 4.8 in the previous chapter, the solution investigated in detail provided good accuracy on types, which had more instances, such as “Normal”, “Neptune”, “Smurf”, but performed badly on others which appeared with less regularity.

Section 5.1: The Notion of Dynamic Fitness

In order to improve the accuracy on the relatively ‘rare’ connection types without compromising performance on the more frequent, a dynamic weighting is introduced. To do so, type specific weights (penalties) will be updated periodically in proportion to the current error rates on different connection types. Since subset selection is used (section 3.1), and current error rates are calculated on the basis of a portion of the training set (the block), then connection type weighting will be introduced in a similar way.

Section 5.2: Implementation and Analysis of Dynamic Fitness

In line with its five generic connection types, “Normal”, “Probe”, “DOS”, “U2R”, “R2L”, the fitness weighting will reflect the same set of categories. On initialization, every category received an initial weight, which can be equal (no a priori information) or biased (a priori information). At every RSS selection, error rates for all categories were

calculated and weights updated accordingly. For example, at iteration *j*, the weight on category *c*, $W_j(c)$ is calculated as an exponential weighted history of previous weight value, or as followings.

$$W_j(c) = \alpha \times E_{j-1}(c) + (1 - \alpha) \times W_{j-1}(c) \qquad (4)$$

Thus as $\alpha \to 1(0)$, more emphasis is given to the most recent error on connection type *c* (previous instances of error). To this end, four variations are considered, Table 5.1. Case 1 was investigated in section 4. Case 2 and 3 consider 'high' (low) emphasis of the correct error estimate, given equal initial weight. The final case initialized in favor of minimizing 'Normal' connection type first, for a 'high' emphasis of the current error estimate.

Table 5.1 Standard setting and three combinations

Case	Description
1	Weight = 1 for all 5 types
2	α = 0.3, initial weight = 0.2, for all the 5 types
3	α = 0.05, initial weight = 0.2, for all the 5 types
4	α = 0.3, initial weight (normal) = 0.6 and 0.1 for the rest 4 types

30 runs on all 4 cases were conducted. Median, first quartile and third quartile were calculated on the false positives and true positives across the 30 results and used to measure performance. FP rate and detection rate were calculated based on these statistical numbers and drawn in the following charts.

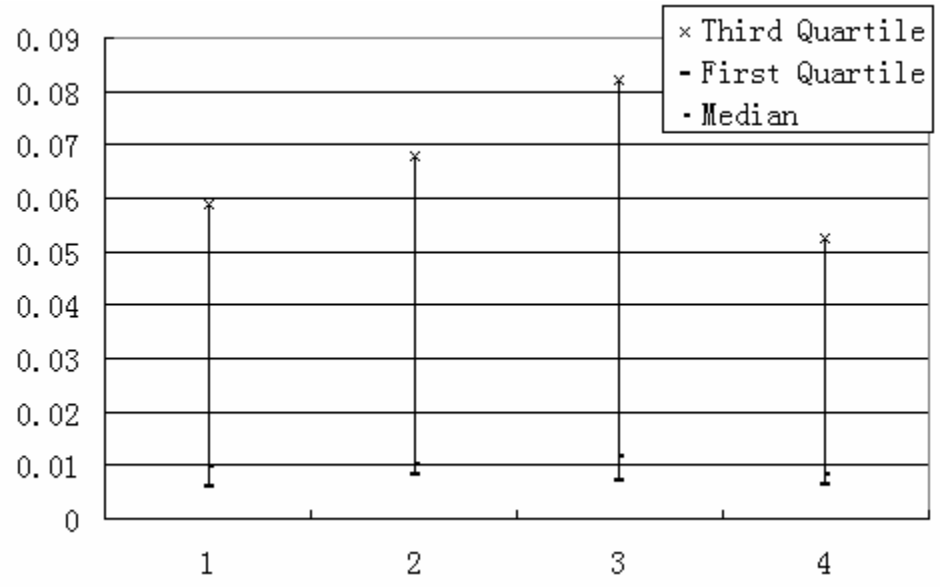

Fig 5.1 FP rates on 4 cases (Y-axis is FP rate.)

In Fig 5.1, case 4, which uses dynamic fitness with initial weight biased to 'Normal', gives best performance on all the statistical numbers (first quartile, median and third quartile) over the other three, in terms of FP rate. And the other two cases using dynamic weights and equal initial weights perform worse than the standard setting, which uses fixed and equal weights. Such results indicate that using biased initial weights plays an important part in "focusing" the training process. In terms of α, difference is apparent that a better median and lower spread is provided for the higher α value.

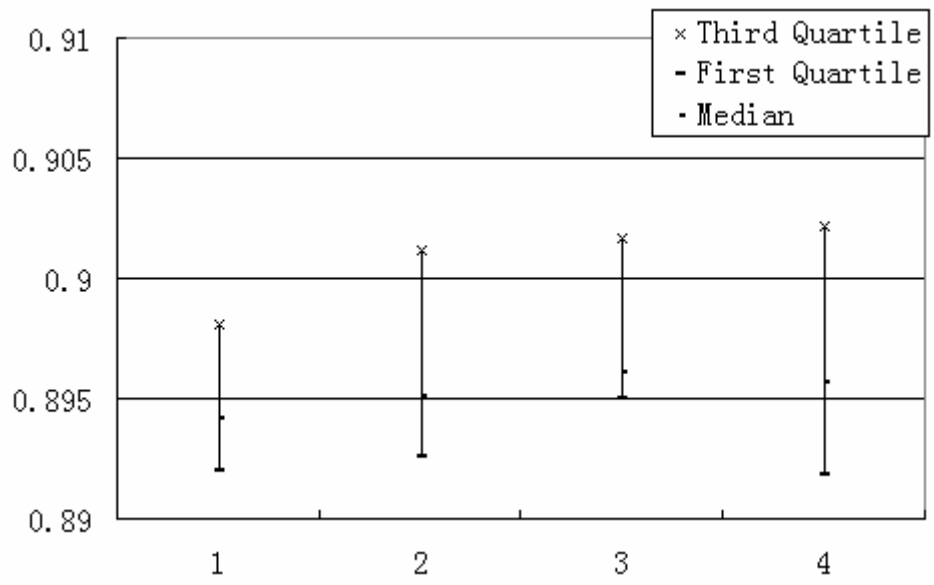

Fig 5.2 Detection Rates on 4 cases. Y-axis is Detection Rate.

In Fig 5.2, all the cases using dynamic weights show the potential ability to improve the detection rate statistics. They all had better third quartile, first quartile and median than the standard settings, except case 4. In terms of α, the smaller value has demonstrates better performance, i.e. case 3 provided less spread than either 2 or 4. Taking into account both of the FP rate and detection rate, case 4 performs best. It has best FP rate as well as an acceptable detection rate (same median as case 3). Weight cases 2 and 3 are considered inferior, even though they are better on the attack side, this is out weighted by much worse 'Normal' or FP rates.

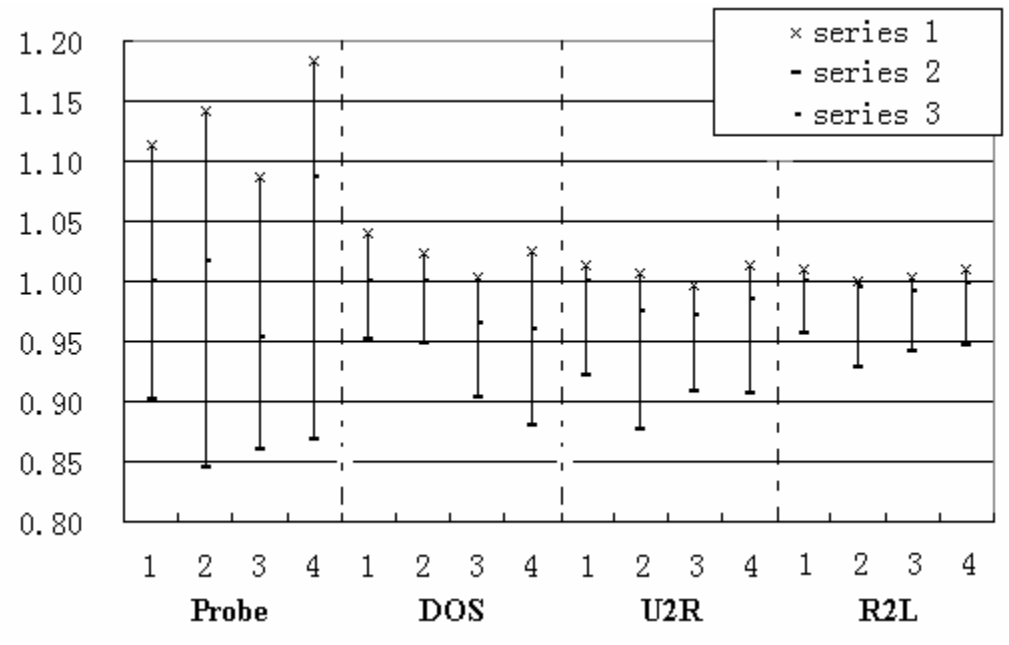

Fig 5.3 Normalized error rates on attack categories from 4 cases. Y-axis is normalized value

In order to investigate whether dynamic fitness (case 2 – 4), compared with case 1, improves the accuracy on connection types with lower frequency of occurrences, error rates on each category from different cases are normalized with respect to the standard fitness (case 1) and plotted into Fig 5.3. Within each attack category ('Probe, 'DOS', 'U2R' and 'R2L'), the normalized value of case *i* in the three series are calculated as followings. The median values from case 1 (standard fitness) in each category served as a baseline, so that both the deviation (between third quartile and first quartile) and average (median) values can be compared between different cases.

$$Series\ 1: \quad v_i = \frac{ThirdQuartile_i}{Median_1} \tag{5}$$

$$Series\ 2: \quad v_i = \frac{FirstQuartile_i}{Median_1} \tag{6}$$

$$Series\ 2: \quad v_i = \frac{Median_i}{Median_1} \tag{7}$$

Thus, in Fig 5.3, lower value means lower error rate. It's apparent that, compared with case 1, all the cases using dynamic fitness have lower error rates on all attack types, especially 'U2R' and 'R2L'. 'U2R' and 'R2L' have the lowest frequency in the training set, respectively, 0.01% and 0.23%. Thus, dynamic fitness not only improves the accuracy on major types, but also on the minor types.

Table 5.2 details classification accuracy on each connection type from best case individual recorded by each fitness weighting – case 1 to 4, Table 5.1 For the best GPs from cases 2 and 3, relative to case 1 (standard fitness), they both perform better on the minor categories, 'U2R' and 'R2L', but are 2 to 3 times worse on 'Normal' connection type. That's a high FP rate, hence they were considered inferior to case 1 and 4. The best GP in case 4 discovered the following rule.

$$Output = 55 - 2 \times Input[0][5] - 3 \times Input[1][5]$$

Compared with the best GP from case 1, it has similar FP rate (0.0375) rate and detection rate (0.904), however as shown in Table 5.2, it performed better on 'R2L'.

Table 5.2 Total number of errors on 5 categories from best GPs from 4 cases

Case	Normal	Probe	DOS	U2R	R2L
1	1979	423	8449	88	14016
2	5838	1111	8079	87	12303
3	6073	593	7441	73	12236
4	2274	1238	9038	182	13686

Chapter 6: Lexicographic Fitness – Motivation and Performance

In chapter 5, dynamic fitness shows the potential to improve the accuracy on connection types with both low and high frequencies. However performance is sensitive to the correct combination of two parameters, α and initial weights, where the best combination might not be obvious initially.

Section 6.1: The Notion of Lexicographic Fitness

To reduce such complexity, lexicographic fitness can serve as a simple alternative formulation for the cost function, which may uphold classification accuracy on major types, like 'normal' and 'DOS' whilst improving that on lesser classes. Specifically, rather than try to formulate a single cost function for all classifications (as in the previous experiments), lexicographic fitness functions define a hierarchy of fitness classes [Huelsbergen, 1998], [Luke, 2002]. Such a scheme results in individuals getting the basics right first (in this case separation of major types from minor types) and only then introduce more specific criterion.

Section 6.2: Implementation and Analysis of Lexicographic Fitness

In this chapter, a two level lexicographic fitness is utilized, as follows. **Level 1 fitness** uses error rates on ‘Normal’ and ‘DOS’ whereas **Level 2 fitness** uses error rates on the remaining categories. After each tournament, the individuals are ranked in terms of their first level fitness. If more than two appear within the same range of level 1

fitness, their level 2 fitness will be used to resolve the tie, i.e. in order to resolve the best set of parents in case where level 1 fitness is in the same range, level 2 fitness is used as the tie breaker. As before, 30 runs are conducted and results are compared with the previous 4 cases in chapter 5 using FP, detection rate and error rates on specific types. Fig 6.1 summarized the distribution of FP rates in which the first four entries 1, 2, 3, 4 denote the previous cases and case 5 represents lexicographic fitness. It is apparent that lexicographic metric provides the most concise distribution of the FP metric.

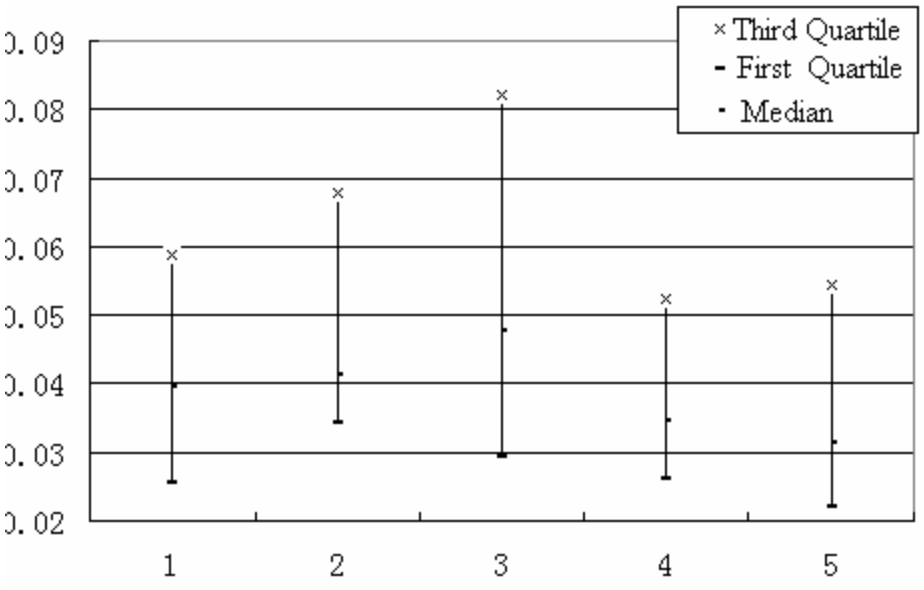

Fig 6.1 FP rate on 5 cases

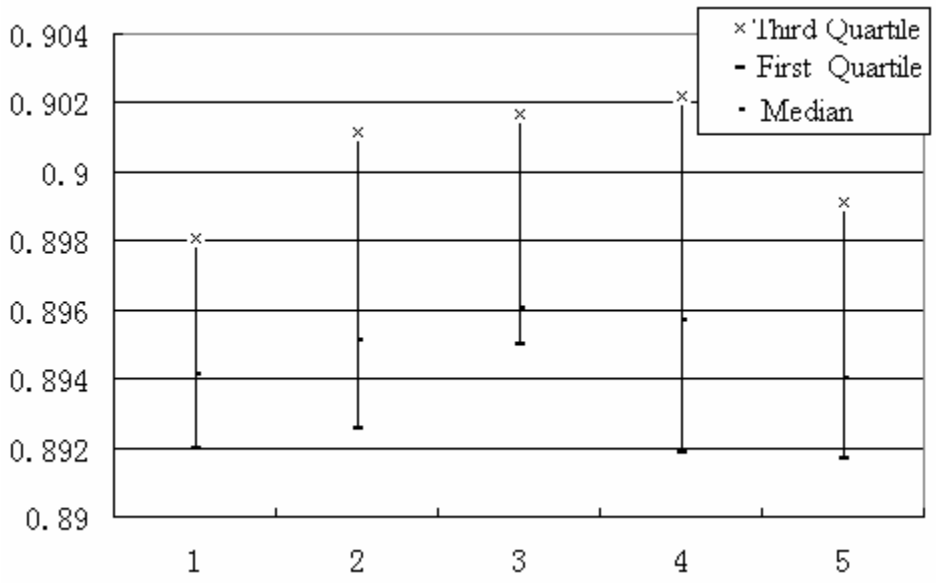

Fig 6.2 Detection rates on 5 cases

In terms of detection rate, as shown in Fig 6.2, lexicographic performs equally well as case 1. Lexicographic has 0.8917 as first quartile, 0.894 as median and 0.899 as third quartile, while case 1 has 0.8919 as first quartile, 0.8941 as median and 0.898 as third quartile. Compared with case 4, which is the best among cases using dynamic fitness, lexicographic still makes comparable accuracy, as shown in Table 6.1. With respect to both of the FP rate and detection rate, lexicographic fitness is superior to all the cases because it achieves outstanding decrease on the FP rate (Fig 6.1) while retaining acceptable accuracy on detection rate.

Table 6.1 Detection rates on case 4 and 5

Quartile	Lexicographic	Experiment 4	Difference
First	0.8917	0.8919	0.0002
Second (Median)	0.894	0.8957	0.0017
Third	0.899	0.9022	0.0032

Regarding accuracy on specific attack categories, the error rates on each type are normalized in the same way in the chapter 5 and plotted in Fig 6.3. Compared with case 1, except on “Probe”, lexicographic has equal or better third quartile and median of error rates over all the attack types. Compared with case 4, lexicographic is inferior on “DOS” types, but has similar accuracy on the remaining types. And in terms of total error rate (total number of errors / total number of patterns), lexicographic has the best performance, as shown in the Fig 6.4. The values are normalized in the same way as in chapter 5.

Moreover, in terms of performance of best GPs from different cases, as shown in Table 6.2, the best GP from lexicographic fitness has the lowest error on “Normal”, and comparable errors on “DOS”. Thus this GP is superior to other cases, as it is low FP rate which will often dictate the usability of the detector.

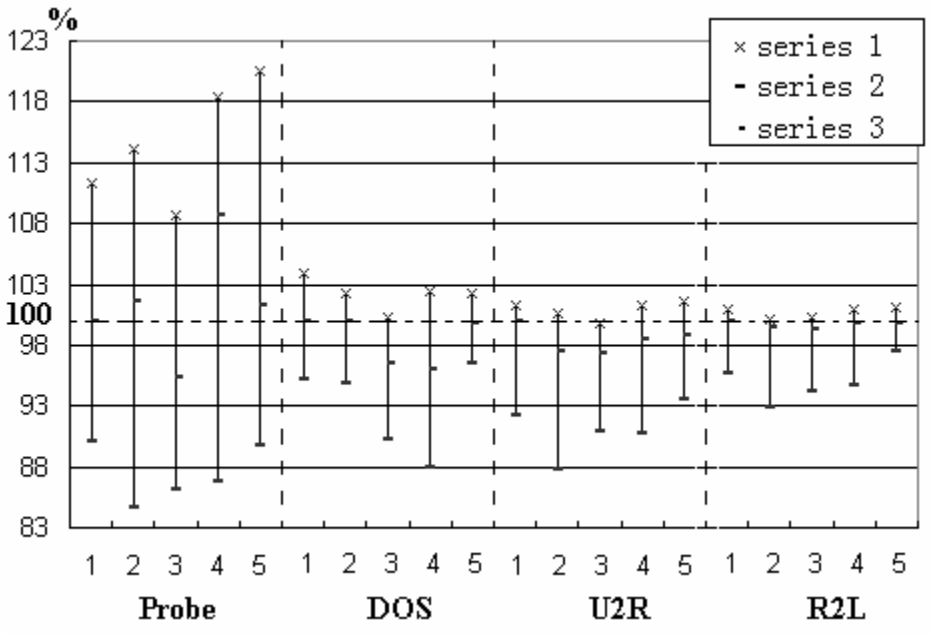

Fig 6.3 Normalized error rates on attack categories

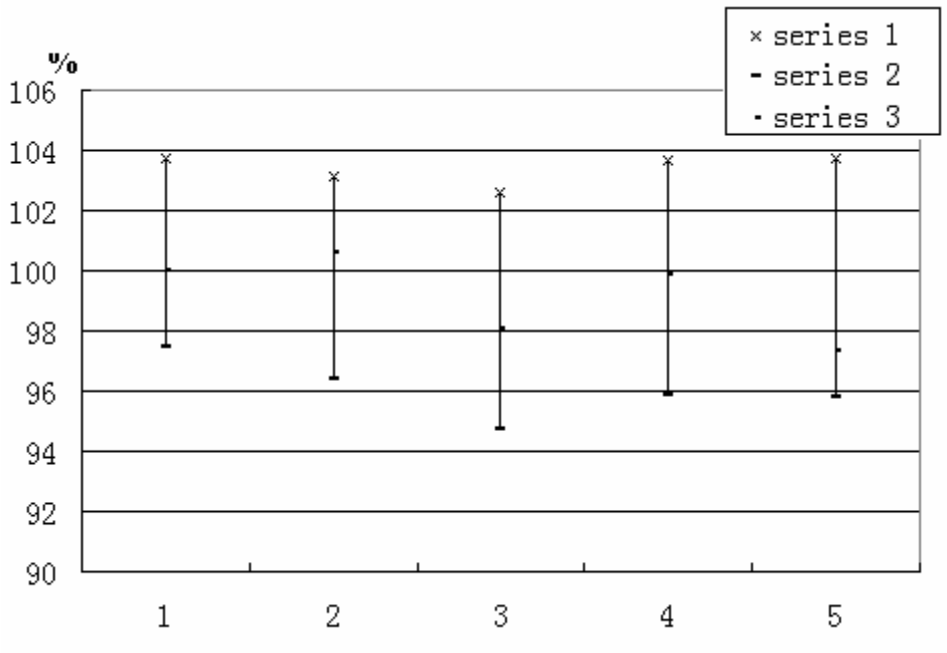

Fig 6.4 Normalized total error rate on 5 cases

Table 6.2 Errors on 5 categories from best GPs

Case	Normal	Probe	DOS	U2R	R2L
Best GP from 1	1979	**423**	8449	88	14016
Best GP from 2	5838	1111	8079	87	12303
Best GP from 3	6073	593	**7441**	**73**	**12236**
Best GP from 4	2274	1238	9038	182	13686
Best GP from 5	**547**	1230	9045	142	14567

Finally, the complexity of this particular individual is much more involved than that found in the previous cases, as per the following relation,

$$Output = R4 \times R0$$

$$R4 = \frac{[5][1]}{[5][6] \times ([5][5] + 1) \times [1][2] \times [1][7]^2}$$

$$R0 \quad = \quad \frac{251 + (2 \times [4][7] + [3][5]) \times [2][4] \times [1][1] + [4][4]}{\frac{[1][7] - 15}{[4][6] \times ([0][0] + 1) \times [4][7]} - [0][4]} + [1][1]$$

where $[m][n]$ represents n^{th} feature in $(t - 4 * m)^{\text{th}}$ connection (Table4.3, Table 4.4) at current record t. Note, however, this should be taken as a general effect of the lexicographic fitness function, but merely a result of the particular individual.

Chapter 7: Conclusion and Future Work

Section 7.1: Summary of Page-based L-GP with Subset Selection

A Page-based Linear Genetic Programming system with Dynamic Sub-Set (DSS) and Random Sub-Set (RSS) schemes for efficient training data sampling was designed, implemented and tested on the KDD'99 benchmark dataset. In doing so, a hierarchy of data subset selections are introduced in order to facilitate learning by GP on a data set of half a million patterns. Moreover, only the basic connection features are employed, with GP deriving the necessary temporal features itself. Table 7.1 demonstrates that GP performance approaches that of data-mining solutions based on all 41 features, whilst solution transparency is also supported and verified, enabling the user to learn from the solutions provided.

Table 7.1 Comparison between KDD Best Entries and Best Lexicographic Solution

Parameter	Detection Rate	FP rate
KDD Winning entry	0.908819	0.004472
KDD Second placed	0.915252	0.00576
Best Lexicographic fitness GP	0.9002	0.00903

Section 7.2: Time Complexity and Solution Complexity

In this work, each GP trial requires approximately 15 minutes to find an optimal computer program to address the two-class classification problem on the KDD'99 dataset. Such solutions take the form of simple, short linear structures of assembly language. In comparison to the solutions previously proposed on the KDD contest

(section 2.2), page based L-GP represents the potential to generate more concise rules in shorter training time, Table 7.2.

Table 7.2 Comparison between KDD Best Entries and Best Lexicographic Solution

Parameter	Training Time Complexity	Solution Complexity
KDD Winning entry	≈24 hours	500 decision trees
KDD Second placed	≈22 hours	755 decision trees
Best Lexicographic fitness GP	≈7.5 hours (30 trials at ≈15min per trial)	Best case L-GP solution with 86 instructions in 2 address format (chapter 3)

Section 7.3: Summary and Comparison on Different Fitness Measurement

Standard, dynamic and lexicographic fitness were designed and tested. In case of a 'standard' fitness measure, equal penalties was assigned to every connection type and maintained at these values during the training process. In dynamic fitness measurement, weights associated with different connection types were adjusted at run time according to their error rate i.e. a multi-objective cost function. In lexicographic fitness, a 2 level fitness function is formed instead of a single cost function for all classifications.

Compared with 'standard' fitness, the other two fitness metrics provide the potential to improve accuracy on minor connection types without compromising the classification ability on major types. In terms of False Positive rates and Detection rates, the lexicographic fitness solution performs best with the principle improvement to FP rate and similar detection rates. The case of dynamic fitness with larger α value and biased initial weight to 'Normal' also appeared to provide an advantage over 'standard' fitness.

The cases using smaller α or equal initial weights proved inferior. The principle reason for this deterioration appeared to be an over emphasis on classifying the less frequent connection types at the expense of the more frequent.

Compared with the best programs from different fitness metrics, the individual utilizing a lexicographic fitness achieves the lowest number of errors on 'Normal', while the others are 4 – 10 times worse, Table 6.2. In terms of the total number of errors on all the attacks, each demonstrate relatively close performance, Table 6.2. Thus, the individual from lexicographic is considered the best solution, while the solutions from standard and case 4 are considered equally good, or be if with the significant overhead of selecting the correct parameter values.

Section 7.4: Future Work

The GP function set in all the reported experiments is purely arithmetic. Of interest would be the significance of introducing conditional statements or modular code within this problem context. Moreover, in this work, a single individual is used to recognize all connection types, in the future; a co-evolution system has the potential to provide individuals that co-operate in any one classification thus providing the basis for distributed or modular solutions; as would the case of building classifiers hierarchically to provide increasingly more specific attack types. Finally, rather than limiting ourselves to the case of the 8 most basic features, GP could also be evolved for that case of all 41 connection features in section 2.1, thus giving a direct comparison between KDD contest best entries and GP solutions.

An interesting fact to note in this work is that, even though no knowledge of application data can be derived from the 8 basic TCP/IP features employed (section 4.1), GP solutions still appear to recognize connections which would be commonly regarded as content based attacks, i.e. they can be only distinguished from the normal traffic by examining the application data carried in the packets. 'Httptunnel', 'Phf' and 'Apache2' in the Table 4.8 are typical such attacks yet the GP individual still recognized 84%, 50% and 35% out of them respectively. Thus, study of rules generated from GP solutions would likely provide a good way to explore unknown abnormal *traffic* patterns useful for identifying content-based attacks.

Reference

[Brameier, 2001] Brameier M., Banzhaf W.: A Comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining. IEEE Transactions on Evolutionary Computation, 5(1) (2001) 17- 26

[Caberera, 2000] Caberera J.B.D., Ravichandran B., Mehra R.K.: Statistical traffic modeling for network intrusion detection. Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (2000) 466 –473

[Cramer, 1985] Cramer N.L.: A Representation for the Adaptive Generation of Simple Sequential Programs. Proceedings of the International Conference on Genetic Algorithms and Their Application (1985) 183-187

[Song, 2003] Song D., Heywood M.I., Zincir-Heywood A.N.: A Linear Genetic Programming approach to Intrusion Detection. GECCO 2003

[Elkan, 2000] Elkan C.: Results of the KDD'99 Classifier Learning Contest. SIGKDD Explorations. ACM SIGKDD. 1(2), (2000) 63-64

[Friedburg, 1958] Friedburg R.M.: A Learning Machine: Part 1. IBM Journal of Research and Development. 2(1), pp 2-13

[Gathercole, 1994] Gathercole C., Ross P.: Dynamic Training Subset Selection for Supervised Learning in Genetic Programming. Parallel Problem Solving from Nature III. Lecture Notes in Computer Science, Vol. 866. Springer-Verlag (1994) 312-321

[Hennessy, Patterson 2002] Hennessy J.L., Patterson D.A., Computer Architecture: A Quantitative Approach, 3rd Edition. Morgan Kaufmann, ISBN 1-55860-569-7 (2002

[Heywood, 2002] Heywood M.I., Zincir-Heywood A.N.: Dynamic Page-Based Linear Genetic Programming. IEEE Transactions on Systems, Man and Cybernetics – PartB: Cybernetics. 32(3) (2002), 380-388

[Holland, 1975] Holland J.H., Adaptation in Natural and Artificial Systems. University of Michigan Press.

[Huelsbergen, 1996] Huelsbergen L., "Toward Simulation Evolution of Machine-Language Iteration," Proceedings of the Conference on Genetic Programming. pp 315-320

[Huelsbergen, 1998] Huelsbergen L., "Finding General Solutions to the Parity Problem by Evolving Machine-Language Representations," Processings of the 3rd Conference on Genetic Programming. Morgan Kaufmann. Pp 158-166

[Kendall, 1998] Kendall K.: A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems. Master Thesis. Massachusetts Institute of Technology (1998).

[Koza 1989] Koza J.R., "Hierarchical Genetic Algorithm Operating on Populations of Computer Programs," Proceedings of the 11th International Joint Conference on Genetic Algorithms. Sridhara N.S. (ed.), Morgan Kaufmann. Pp 768-774.

[Koza, 1992] Koza J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA, MIT Press (1992)

[Koza, 1999] Koza J.R., Bennett F.H., Andre D., Keane M.A., Genetic Programming III: Darwinian Invention and Problem Solving, Morgan Kaufmann, ISBN 1-55860-543-6 (1999).

[Levin, 2000] Levin I.: KDD-99 Classifier Learning Contest LLSoft's Results Overview. SIGKDD Explorations. ACM SIGKDD. 1(2) (2000) 67-75

[Lippmann, 2000] Lippmann R.P., Fried D.J., Graf I., Haines J.W., Kendall K.R., McClung D., Weber D., Webster S.E., Wyschogrod D., Cunningham R.K., Zissman M.A.: Evaluating Intrusion Detection Systems: the 1998 DARPA Off-Line Intrusion Detection Evaluation. Proceedings of the 2000 DARPA Information Survivability Conference and Exposition, 2 (2000)

[Luke, 2002] Luke S., Panait L., "Lexicographic Parsimony Pressure," Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Langdon W.B., et al., (eds.), Morgan Kaufmann. pp 829-836.

[McHugh, 2000] McHugh J.: Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory. ACM Transactions on Information and System Security. 3(4), (2000) 262–294

[Nordin, 1994] Nordin P., "A Compiling Genetic Programming System that Directly Manipulates the Machine Code," in Advances in Genetic

Programming. Kinnear K.E. (ed.), Chapter 14. MIT Press, pp 311-334.

[Pfahringer, 2000] Pfahringer B.: Winning the KDD99 Classification Cup: Bagged Boosting. SIGKDD Explorations. ACM SIGKDD. 1(2) (2000) 65-66

[Poli, 2001] Poli R., "Exact Schema Theory for Genetic Programming and Cariable-Length Genetic Algorithms with One-Point Crossover," Genetic Programming and Evolvable Machines. 2(2), pp 123-165.

[Syswerda, 1991] Syswerda G., "A Study of Reproduction in Generational and Steady-State Genetic Algorithms," in Foundations of Genetic Algorithms. Rawlins G.J.E. (ed.), Morgan Kaufmann, pp 94-101

[Vladimir, 2000] Vladimir M., Alexei V., Ivan S.: The MP13 Approach to the KDD'99 Classifier Learning Contest. SIGKDD Explorations. ACM SIGKDD. 1(2) (2000) 76-77

[Wenke, 1999] Wenke L., Stolfo S.J., Mok K.W.: A data mining framework for building intrusion detection models. Proceedings of the 1999 IEEE Symposium on Security and Privacy (1999) 120 –132

[Wilson, 2002] Wilson G.C., Heywood M.I., "Crossover Context in Page-Based Linear Genetic Programming," Canadian Journal of Electronic and Computer Engineering. 27(3), pp 113-116.