Cascaded GP Models for Data Mining

Peter Lichodzijewski
Dalhousie University
Faculty of Computer Science
6050 University Avenue, NS.
B3H 1W5, Canada
Email: piotr@cs.dal.ca

Malcolm I. Heywood
Dalhousie University
Faculty of Computer Science
6050 University Avenue, NS.
B3H 1W5, Canada
Email: mheywood@cs.dal.ca

A. Nur Zincir-Heywood
Dalhousie University
Faculty of Computer Science
6050 University Avenue, NS.
B3H 1W5, Canada
Email: zincir@cs.dal.ca

***Abstract*-The cascade architecture for incremental learning is demonstrated within the context of genetic programming. Such a scheme provides the basis for building steadily more complex models until a desired degree of accuracy is reached. The architecture is demonstrated for several data mining datasets. Efficient training on standard computing platforms is retained using the RSS-DSS algorithm for stochastically sampling datasets in proportion to exemplar 'difficulty' and 'age'. Finally, the ensuing empirical study provides the basis for recommending the utility of sum square cost functions in the datasets considered.**

I. Introduction

The inner loop of genetic programming (GP) represents a significant impediment to the utilization of the approach in data mining problems. There are two aspects to this problem. The first is that the scaling of GP to multi-class problems typically requires a separate population for each class [1]. The second is the large number of exemplars which typify data mining problems. In this work, we are interested in the latter. Moreover, we are also interested in returning concise but accurate results.

Traditionally the approach used for addressing the inner loop problem is through parallel processing. Specific examples include hardware acceleration of the fitness evaluation [2], special purpose parallel machines [3], or more recently the use of Beowulf clusters [4], [5]. In this work, use is made of a rather different approach. No assumption is made regarding the hardware, but use is made of the observation that not all exemplars are created equal. That is to say, candidate solutions are used to filter the dataset so that the most difficult or oldest exemplars are favoured. Such a scheme was recently demonstrated on the KDD-99 data mining competition dataset, with trails being completed in 15 minutes for a dataset consisting of half a million exemplars [6]. It was denoted the RSS-DSS algorithm and is applicable to a broad range of machine learning algorithms.

In this work, we utilize the RSS-DSS algorithm to build classifiers with a greater degree of accuracy than was previously the case. To do so, use is made of the Cascade Correlation Architecture originally proposed within the context of neural networks [7]. Such an approach provides the basis for incrementally building increasingly sophisticated classifiers. In doing so, we provide an alternative approach to addressing the problem of model complexity, that is to say, variable length individuals has been the norm with GP. This avoids the need for arbitrary design decisions regarding the likely complexity of the solution, but code bloat and less than opaque solutions are often the resulting penalty.

This work demonstrates that it is now feasible to incrementally build concise accurate solutions to data mining problems using GP. The proposed model is demonstrated on the Adult dataset (≈30000 training exemplars) and an Intrusion Detection dataset (≈66000 training exemplars). The RSS-DSS algorithm is used to provide the efficiency behind what would normally be a computationally expensive approach to building hierarchical models. Classification performance is demonstrated to improve on that provided by a recently proposed parallel GP model with boosting [5]. Experiments are also performed with a range of different cost functions. In particular, the GP literature has typically dealt with classification problems by assuming a cost function based on the count of correct classifications. In this work, we also investigate the case of absolute, sum square error and Bernoulli cost functions.

The paper is organized as follows. Section II provides the methodology for the proposed cascaded GP. Section III summarizes the RSS-DSS algorithm used to filter the dataset during training, thus minimizing the computational overhead of the inner loop. Section IV details the fixed length linear GP employed during these experiments, although there is nothing in the cascade model for building models which is specific to this. Section V presents the results over a range of different cost functions and in doing so establishes the case for a sum square error cost. Conclusions are drawn in Section VI.

II. Cascade Architecture

The cascade architecture investigated by this work is based on the method presented in [7], which in turn is an adaptation of the work in [8]. Figure 1 summarizes the basic algorithm. Each iteration of the outermost loop yields one layer in the cascade. It is apparent that layers are built incrementally with the GP output (GPout) from the best individual at each layer augmenting the dataset. Thus, layer 'l' receives an exemplar composed of the original features plus the output from the $l - 1$ previous layers on the same exemplar. This is synonymous with each model layer 'seeing' all the outputs from models built at previous layers in addition to the original input exemplar as defined by the dataset. Best case classifiers are identified at each layer as the result of a competition across 'N' populations (10 for the experiments performed herein), thus minimizing the significance of a specific initialization.

```
do
{
        initialize 'N' populations;
        for (i < N) train pop(i);
        best = fittest{pop(0), ..., pop(N-1)}
        cat(Data, GPout(best));
}
until (error criterion satisfied);
```

Figure 1. Cascade GP algorithm.

A. Cost Function and Wrapper

Classification problems typically require a wrapper for reformulating the GP output in terms of a specific class. Typical practice has been to employ a switching function of the form "If (GPout > 0) THEN (class = 1) ELSE (class = 0)". Such a scheme precludes the utilization of sensitivity analysis (e.g., ROC curves) or the provision of any certainty information regarding the class label provided. In addition such a switching wrapper precludes the use of alternative cost functions such as the Bernoulli error, where this is explicitly formulated for the case of binary objectives as encountered in classification problems [9].

To this end, a total of four cost functions are considered. These are shown in Table I where y corresponds to GP output, p corresponds to a pattern, and d corresponds to the desired output. These functions are,

Error Count (EC): This is the hits-based metric typically employed for GP classification problems. The wrapper naturally takes the form of the switching function;

Absolute Error (ABS): This is merely the absolute difference between desired and required classification. The basic implication of such a cost function is that all errors are penalized equally. The accompanying wrapper takes the form of a squashing function mapping the entire real number line to the interval +/- 1.0. Desired values now take the form +/- 0.8, with the objective of encouraging GP not to produce saturated output values;

Sum Square Error (SSE): This cost function results in errors larger than unity receiving more penalty than those below, c.f. the square law. Moreover, such a formulation assumes that errors follow a normal distribution. The wrapper and desired values remain the same;

Bernoulli Error (BE): Classification problems typically express the desired values using a binary label. As a consequence, limiting the model output to continuous values over the unit interval results in a probabilistic formulation of the cost function, or a Bernoulli Error [9]. Such a cost function provides a penalty function with a sharp knee. The matching GP wrapper naturally maps the real number line to the interval [0, 1].

TABLE I
COST FUNCTIONS AND CORRESPONDING WRAPPER

Wrapper	Cost function
IF ($y > 0$) THEN (y = 1) ELSE ($y = 0$)	EC = $\sum_p(1 - \text{hit}(p))$ where hit = 1 IF $y == d$ ELSE hit = 0
$2 \times (1 + \exp(-y))^{-1} - 1$	ABS = $\sum_p \text{abs}(d_p - y_p)$ SSE = $\sum_p (d_p - y_p)^2$
$(1 + \exp(-y))^{-1}$	BE = $\sum_p -\ln(d_p\, y_p - (1 - y_p)(1 - d_p))$

III. RSS-DSS ALGORITHM

The RSS-DSS algorithm is used to train each of the N GP populations at each layer in the cascade architecture. As indicated in the introduction, this algorithm is utilized for filtering large datasets such that GP is only trained on a fraction of the dataset at any one time [6]. To do so, two basic concepts are employed. Firstly, exemplars should be selected stochastically relative to their 'difficulty' and 'age' [10]. That is, exemplars that are older or more difficult should have a greater chance of being selected. Secondly, the original dataset is partitioned into blocks that are compatible with the available memory hierarchy.

The RSS-DSS hierarchy has two layers. At the first layer, training data is partitioned into blocks that can fit into memory. During training, these blocks are randomly selected with uniform probability. Each selected block has an associated history of training pressure (based on the error rate on that block on its previous selection) and this history is used to determine the number of iterations at the next layer (i.e., more difficult blocks are trained for more iterations).

At the second layer, a subset of exemplars is selected from the first-layer block. Each selection into this subset is made based on the exemplars' difficulty or age, and for each selection, the choice of which value to use is based on a fixed probability. Fitness evaluation is then conducted on this subset drastically reducing the overhead associated with the innermost loop of GP. After a fixed number of tournaments, the subset is reselected.

More specifically, the age of an exemplar is the number of DSS selections since it was last selected, and its difficulty is the number of individuals that were unable to recognize it correctly the last time that it appeared in the DSS subset. Roulette wheel selection with respect to the age or difficulty is used for the subset selection. Once the selection is complete, the age and difficulty of selected exemplars is reset, while for all other exemplars age is incremented by one and the difficulty is left as is.

IV. PAGE-BASED LINEAR GP

Linearly structured genetic programming (L-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. 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 general-purpose 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 exemplar). In this work, a 2-address instruction format is employed. 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 and not through the opcode.

As with the case of tree-structured GP many instances of L-GP have been developed over a considerable period of time [12-15]. The emphasis of this work however, is the cascade algorithm, from which the only requirement is that the selection operator be a steady-state tournament. The specific form of L-GP employed by this work utilizes the page-based L-GP developed in an earlier work [16]. Such a scheme is fixed length. Its basic components are as follows.

Representation: Individuals take the form of a 2-address instruction format. Individuals are of fixed length described in terms of a (uniformly) randomly selected number of pages, where each page has a fixed number of instructions.

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, …, maxPages]. That is to say the initial population is initialized over the range [min program length, …, max program length]. 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). Secondly, 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.

Selection Operators: The RSS-DSS algorithm requires a steady-state tournament. In this case all such tournaments are conducted with four individuals. The two fittest individuals are retained and reproduce. The children over-write the worst two individuals from the same tournament.

Search Operators: Three search operators are utilized, each with a corresponding probability of application, where such tests are applied inclusively (i.e., the resulting children might be the result of all three search 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 pages. Mutation has two forms. The first case – hereafter referred to as 'mutation' – merely Ex-OR's an instruction with a new instruction and confirms that the resulting instruction decodes into a valid instruction. (No benefits were observed in making such a mutation operator 'field specific', where this is undoubtedly a factor of the addressing format [16].) 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 posses 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 an algorithm was introduced for modifying the number of instructions per page dynamically during the course of the training cycle [16]. In this case the user defines the maximum page size as an order of two. The page size is then doubled for each plateau in the fitness function, beginning with a page size of one and finishing at 'max page size' and returning to a page size of one once a plateau following 'max page size' is encountered. Plateaus are defined in terms of consecutive non-overlapping windows of ten tournaments. For each of the ten tournaments the (tournament's) best-case individual's fitness is summed. If the sum 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 (2 boxes, 6 parity, UCI classification benchmarks) [16]. Moreover, solutions also tended to be more concise than the alternative tree-structured and variable length L-GP.

We emphasize, however, that the principle interest of this paper lies in the cascade algorithm, which is equally applicable to any form of GP.

V. Results

Two datasets were used to evaluate the architecture. The first was the Adult dataset obtained from the UCI Machine Learning Repository [17]. The version used had unknown instances removed yielding 30162 training patterns and 15060 test patterns. Moreover, this dataset has been used as part of the evaluation of a parallel GP model based on a partitioned boosting algorithm [5].

The second dataset, derived from the seven weeks of training data used in the 1998 DARPA Intrusion Detection evaluation [18], was the Intrusion Detection dataset. It consisted of TCP connections only. Bro was used to process the network traffic yielding four features for each connection: the service used (e.g., telnet, http), the duration, the number of bytes sent by the source of the connection, and a Bro flag indicating the state of the connection [19]. The seven weeks' worth of data was partitioned into training and test data, with three-quarters of the original data used for training. The training data was balanced so that the number of

positive and negative examples was roughly the same and duplicates were removed from the test data. This resulted in 66226 training and 17012 test patterns. Sequencing information was encoded by way of a tapped shift register structure [11]. A total of 8 taps at intervals of 4 were utilized meaning that in addition to the current connection GP could access features spanning the previous 31 connections.

Basic parameterization of GP and RSS-DSS is summarized in Table II. A total of 16 layers are built using the cascade algorithm for each of the four cost functions.

A. Adult Dataset

Table III summarizes the error on the training and test sets of the Adult dataset for layers which showed unique and improved performance. (Note that test data is not used to adapt the algorithm.) Also shown is the length of the best individual at each layer.

TABLE II
LINEAR GP – RSS-DSS PARAMETERIZATION

Parameter	Value
Population Size	125
Max. No. of Pages	32 Pages
Max Instructions Per Page	8 Instructions
Crossover Prob.	0.9
Mutation Prob.	0.5
Swap Prob.	0.9
Tournament Size	4
Function Set	{+, -, *, /}
Terminal Set	{0, …, 255} ⋃ {Features}
RSS Subset Size	5000
DSS Subset Size	50
RSS Iterations	1000
DSS Iterations	100
DSS Subset Refresh Rate	6 Tournaments
Prob. of DSS Selection Based on Age	0.3

Based on the test error, the function that performed best was the SSE function with an error rate of 15.73%. Surprisingly because it was the simplest, the function that performed second best was the EC function. The worst performer was the ABS function with an error rate of 16.39% on the test data.

When building layers using the cascade algorithm, periods during which no change in the classification accuracy were observed, where the characteristics of this property varied as a function of the cost function employed. The ABS function appeared to build a sequence of unique layers early on but failed to do so after layer 7. The EC function exhibited the opposite behaviour, building unique layers towards the end, although the total improvement at that point was not as large. The BE function tended to make consistent progress before adding several layers which had no impact and then added several other layers before 'maxing out'. The SSE function seemed to make steady progress until layer 12. Ideally, a function should not plateau like the ABS function but should continue to improve like the SSE function.

One reason for the plateau in performance was that at times layers would simply forward the output of a previous layer. Translating an individual to human-readable form showed that in such instances (often) two-instruction individuals would load a constant and divide it by a previous output, merely copying the sign of that output and not really contributing any towards an improvement. One possible means of avoiding this behaviour may be to employ concepts from niching to encourage diverse behaviour between the layers.

TABLE III
ADULT TRAINING AND TEST % ERROR PER LAYER

Function	Layer	Train Error	Test Error	Length
EC	0	16.80	16.89	23
	1	16.53	16.52	9
	5	16.09	16.23	43
	8	16.20	16.18	2
	10	15.91	15.99	29
	11	15.80	15.84	4
	12	15.77	15.82	7
ABS	0	17.37	17.36	50
	1	16.98	17.15	62
	2	16.91	17.08	16
	3	16.44	16.65	28
	5	16.30	16.46	31
	7	16.23	16.39	37
SSE	0	16.13	16.25	75
	2	15.73	15.82	27
	4	15.71	15.78	51
	6	15.69	15.94	57
	8	15.66	15.95	24
	11	15.63	15.80	23
	12	15.55	15.73	9
BE	0	16.40	16.61	27
	1	16.19	16.40	61
	2	16.10	16.20	23
	3	16.07	16.18	72
	5	16.05	16.05	63
	6	15.96	16.09	12
	7	15.91	16.03	29
	10	15.90	16.03	69
	15	15.88	16.02	28

Figures 2 and 3 show the false positive and false negative rate for each of the layers in Table III on the Adult training and test data. Trajectories are used to indicate how each layer alters the respective false positive and false negative rate. The superiority of the SSE function is now more apparent as it is able to dominate the lower left-hand side of the plot. Also of interest are the high false positive rates. This is a factor of the unbalanced nature of the dataset where roughly 75% of the dataset belongs to one class.

Relative to previous works, results were available for the Adult dataset using a cellular GP designed to make use of Beowulf computing clusters [5]. In addition to the basic cellular GP, experiments were also reported for

a partitioned implementation in which the dataset was divided into blocks of 5,000 exemplars (as per RSS-DSS). However, each population only saw a single partition. The partitioned variant of the boosting algorithm was then utilized to rebuild the overall classification as an ensemble classifier [20]. Finally, the case of inter processor and no inter processor communication was considered. Table IV summarizes these results. The availability of parallel hardware provides the basis for a very efficient evolutionary cycle. Classification rates, however, are worse than those for the cascade architecture. Naturally, parallelizing the ten populations evolved at each layer of the cascade algorithm would also result in a significant computational improvement.

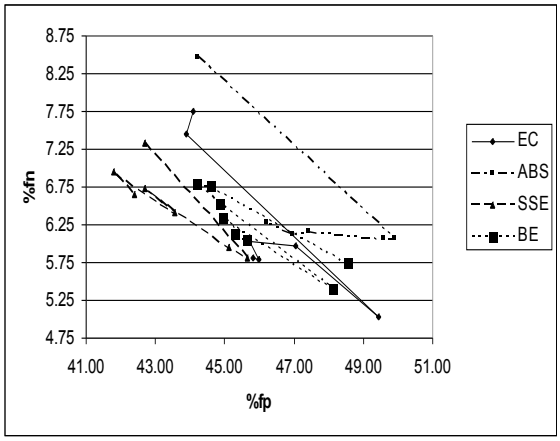

Figure 2. Adult training data false positives and false negatives.

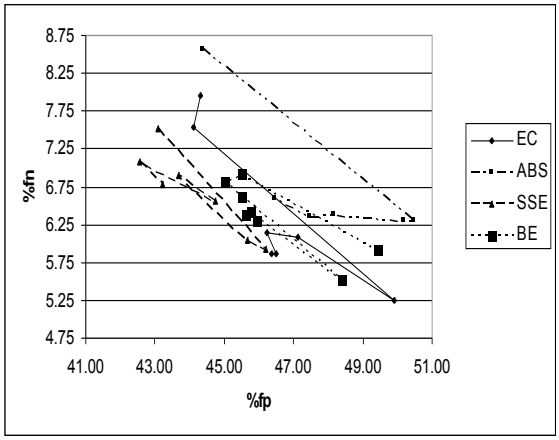

Figure 3. Adult test data false positives and false negatives.

The cascade experiments were run on an iMac desktop with an 800 MHz G4 processor and 1 GB of RAM. The time required to train one layer was roughly three hours or about twenty minutes per run (i.e., one population). In general, the time to train was less at higher layers since the additional features present in each exemplar made the dataset easier. Overall, for all sixteen layers, the total time to evolve a solution was under two days. Although this may not seem as exceptional, it is very reasonable, especially considering that it would not be practical to do the same (i.e., evolve 160 populations) in a similar environment using a GP implementation without the RSS-DSS algorithm.

For comparison purposes, a series of experiments involving Koza's original tree-structured GP were conducted [21]. The experiments used lilgp, an efficient C implementation of tree-structured GP [22]. The parameters used were as follows: population size of 4000, crossover probability of 90%, reproduction probability of 10%, and mutation probability of 0%. As with the page-based L-GP, program size was restricted by a 256-node limit (as opposed to a depth limit). In addition to a base case, a second case that used three Automatically Defined Functions (ADFs) was considered. All experiments were performed on a 1.25 GHz G4 processor with 1GB of RAM which is a faster machine than that used for the training of the cascade architecture.

Table V shows the results over 10 runs for the base case and 8 runs for the ADF case (using ADFs significantly increased run times). Shown are the median (and in brackets, best) values with respect to the training performance. It is now apparent that the cascade algorithm using RSS-DSS returns a significant improvement in classification accuracy. Although the training time is shorter for the individual lilgp runs, had lilgp been used in the cascade architecture training the 160 layers would take roughly 2500 hours.

B. Intrusion Detection Dataset

The original motivation for this work lies in the application of GP to Intrusion Detection problems. As indicated above the DARPA 1998 dataset was preprocessed to provide the basis features for the detector. Table VI shows the results for the Intrusion Detection dataset.

Again based on the test error, the best performer in this case is the ABS function with an error rate of 5.13%. The SSE and BE functions are both very close and come next, and the worst performer is the EC function with a test performance of 5.73%.

Figures 4 and 5 show the trajectories of the false positive and false negative rates from layer to layer. Unlike with the Adult dataset, there is really no one function that dominates the lower left-hand side of the plot. Because of this, and because it performed best on the more difficult Adult dataset, the optimal function appears to be SSE.

TABLE IV
PARALLEL CELLULAR GP TEST SET RESULTS ON ADULT DATASET

Algorithm	% Test Error
Parallel Cellular GP (PCGP)	17.26
PCGP + Boosting	18.26
PCGP + Boosting + communication	16.58

TABLE V
TREE-STRUCTURED GP ON ADULT DATASET, NO RSS-DSS OR CASCADE ALGORITHM

Algorithm	% Train Correct	% Test Correct	CPU Time (Hrs)
No ADFs	82.95 (83.33)	82.91 (83.15)	16.3 (15.74)
3 ADFs	73.4 (80.43)	71.7 (80.62)	25.2 (28.47)

The time require to train on the Intrusion Detection dataset was about half of that required to train the Adult dataset or about ten minutes per run. This is despite the Intrusion Detection dataset being roughly twice as large as the Adult dataset. The reason for this is that the Adult dataset is more difficult, meaning that the RSS-DSS algorithm cannot use the speedup which relies on reducing the number of DSS iterations for blocks that are found to be easier. This reinforces that a major factor in the running time of the RSS-DSS algorithm is the difficulty of underlying dataset.

VI. CONCLUSION

An approach for the automated incremental building of GP solutions using cascade architectures is proposed. Although the cascade approach was used with GP, it can be applied to any learning paradigm.

By combining the scheme with the RSS-DSS algorithm the total computation time on a single computer is acceptable whilst classification rates are also very competitive, significantly bettering that available from a single GP classifier and matching that provided by multi-population models. Moreover, parallelization of the multi-population step will naturally lead to a significant speedup (depending on the dataset an entire population is trained in 10-20 minutes over 30,000 to 66,000 exemplars). Finally, a preference is demonstrated for a sum square error cost on the benchmarks investigated since it clearly dominated when implemented on a more difficult dataset.

Future work will consider the utilization of niching algorithms for establishing less overlap in the operation of the previous and current layers as well as an analysis of the form of the programs evolved at different layers.

TABLE VI
INTRUSION DETECTION TRAINING AND TEST % ERROR PER LAYER

Function	Layer	Train Error	Test Error	Length
EC	0	4.31	6.41	39
	1	3.85	5.73	27
ABS	0	4.57	6.70	45
	1	4.50	6.65	58
	2	3.60	5.36	48
	6	3.47	5.15	35
	13	3.46	5.13	19
SSE	0	6.53	7.37	75
	1	6.52	7.37	27
	2	5.74	6.24	51
	3	3.71	5.60	57
	4	3.59	5.36	24
BE	0	6.57	7.48	17
	1	4.74	6.90	21
	2	4.31	6.37	24
	4	4.29	6.32	14
	6	3.72	5.63	79
	10	3.56	5.33	31

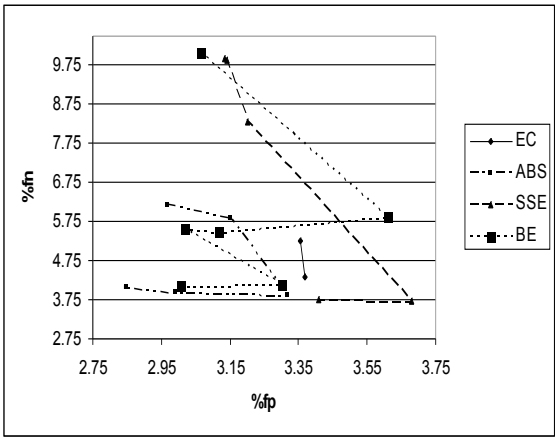

Figure 4. Intrusion Detection training data false positives and false negatives.

ACKNOWLEDGEMENTS

This work was conducted while Peter Lichodzijewski was funded by a NSERC PGS-A scholarship. Malcolm I. Heywood and A. Nur Zincir-Heywood would like to acknowledge the support of NSERC and CFI in conducting this research.

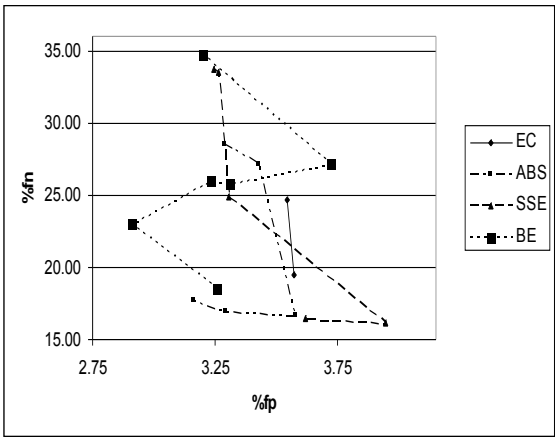

Figure 5. Intrusion Detection test data false positives and false negatives.

REFERENCES

[1] J. K. Kishore, L. M. Patnaik, V. Mani, V. K. Agrawal, "Application of genetic programming for multicategory pattern classification*", IEEE Transactions on Evolutionary Computation*, 3(4), pp. 242-258, 2000.

[2] J. R. Koza et al., "Evolving computer programs using reconfigurable gate arrays and genetic programming," in *Proceedings of the ACM 6th International Symposium on Field Programmable Gate Arrays*, 1998, pp. 209-219.

[3] H. Juillé, J. B. Pollack, "Massively parallel genetic programming," in *Advances in Genetic Programming*, 2nd ed., P. J. Angeline, K. E. Kinnear, Eds. Cambridge, MA: MIT Press, 1996, pp. 339-358.

[4] F. H Bennett III et al., "Building a parallel computer system for $18,000 that performs a half petra-flop per day," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 1999, pp. 1484-1490.

[5] G. Folino, C. Pizzuti, G. Spezzano, "Ensemble techniques for parallel genetic programming based classifiers," in *Proceedings of the 6th European Conference on Genetic Programming*, 2003, pp. 59-69.

[6] D. Song, M. I. Heywood, A. N. Zincir-Heywood, "A linear genetic programming approach to intrusion detection," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2003, pp. 2325-2236.

[7] S. E. Fahlman, C. Labiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems*, 2nd ed., D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 524-532.

[8] E. Littmann, H. Ritter, "Cascade network architectures," in *Proceedings of the International Joint Conference on Neural Networks*, 1992, pp. 398-404.

[9] P. J. Werbos, "Links between artificial neural networks (ANN) and statistical pattern recognition," in *Artificial Neural Networks and Statistical Pattern Recognition: Old and New Connections*, I. K. Sethi, A. K. Jain, Eds. Elsevier Science, 1991, pp. 11-31.

[10] C. Gathercole, P. Ross, "Dynamics training subset selection for supervised learning in genetic programming," in *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, 1994, pp. 312-321.

[11] D. Song, "A linear genetic programming approach to intrusion detection," Masters Thesis, Dalhousie University, Faculty of Computer Science, 2003.

[12] R. M. Friedberg, "A learning machine: Part I," *IBM Journal of Research and Development*, 2(1), pp. 2-13, 1958.

[13] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs, " in *Proceedings of the International Conference on Genetic Algorithms and Their Application*, 1985, pp. 183-187.

[14] L. Huelsbergen, "Finding general solutions to the parity problem by evolving machine-language representations, " in *Proceedings of the 3rd Conference on Genetic Programming*, 1998, pp. 158-166.

[15] 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, pp. 311-334.

[16] M. I. Heywood, A. N. Zincir-Heywood, "Dynamics page-based linear genetic programming," *IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics*, 32(3), pp. 380-388, 2002.

[17] UCI Machine Learning Repository. [Online] http://www.ics.uci.edu/~mlearn/MLRepository.html. (Accessed December, 2003)

[18] 1998 DARPA Intrusion Detection Evaluation. [Online] http://www.ll.mit.edu/IST/ideval/. (Accessed May, 2003)

[19] Bro. [Online]. http://www.icir.org/vern/bro-info.html. (Accessed May, 2003)

[20] L. Breiman, "Pasting small votes for classification in large databases and on-line," *Machine Learning*, 36(1/2), pp. 85-103, 1999.

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

[22] B. Punch, E. Goodman. lilgp Genetic Programming System, v 1.1. [Online]. http://garage.cps.msu.edu/software/lil-gp/lilgp-index.html. (Accessed January, 2004)