Towards Efficient Training on Large Datasets for Genetic Programming

Robert Curry and Malcolm Heywood

Dalhousie University, Faculty of Computer Science, 6050 University Avenue, Halifax, Nova Scotia, Canada, B3H 1W5
{rcurry, mheywood}@cs.dal.ca

Abstract. Genetic programming (GP) has the potential to provide unique solutions to a wide range of supervised learning problems. The technique, however, does suffer from a widely acknowledged computational overhead. As a consequence applications of GP are often confined to datasets consisting of hundreds of training exemplars as opposed to tens of thousands of exemplars, thus limiting the widespread applicability of the approach. In this work we propose and thoroughly investigate a data sub-sampling algorithm – hierarchical dynamic subset selection – that filters the initial training dataset in parallel with the learning process. The motivation being to focus the GP training on the most difficult or least recently visited exemplars. To do so, we build on the dynamic sub-set selection algorithm of Gathercole and extend it into a hierarchy of subset selections, thus matching the concept of a memory hierarchy supported in modern computers. Such an approach provides for the training of GP solutions to data sets with hundreds of thousands of exemplars in tens of minutes whilst matching the classification accuracies of more classical approaches.

1 Introduction

The interest of this work lies in providing a framework for efficiently training genetic programming (GP) on large datasets. There are at least two aspects to this problem: the cost of fitness evaluation and the overhead in managing datasets that do not reside within RAM alone. The computational overhead associated with the inner loop of GP fitness evaluation has been widely recognized. The traditional approach for addressing this problem has been hardware based. Examples include Beowulf clusters [1], parallel computers [2] and FPGA solutions [3]. In this work we propose to address the problem through the following two observations. Firstly, within the context of supervised learning, the significance of data sampling algorithms have been widely acknowledged albeit with the motivation to improve error performance, e.g. boosting and bagging [4, 5]. Secondly, memory hierarchies are widely used in CPU architectures, where such hierarchies are based on the concept of temporal and spatial locality [6]. The motivation used here, however, is that any learning algorithm need only see a subset of the total dataset, where the sampling process used to identify such a subset of exemplars should also be sympathetic to the memory hierarchy of the computing platform.

To address these issues the method of Dynamic Subset Selection [7] was revisited and extended to a hierarchy of subset selections. Such a scheme was applied to the 10% KDD-99 benchmark, a dataset consisting of approximately half a million patterns [8]. The dataset was first partitioned into blocks that were sufficiently small to reside within RAM alone. Blocks were then chosen from this partition based on Random Subset Selection (RSS). This forms the level 1 of the selection hierarchy. At level 2, the method of Dynamic Subset Selection (DSS) was used to stochastically select patterns from the block identified at the first level. Several rounds of DSS are performed per level 1 block, with exemplars identified on the basis of their difficulty and age. This hierarchy shall be referred to as a RSS-DSS hierarchy.

In this work we concentrate on the parameterization of the algorithm and extending the evaluation to another dataset (the work of Song *et al.*, concentrated on application issues associated with the application to an intrusion detection problem [8]). Furthermore, an alternative hierarchy is introduced in which the level 1 block selection was also chosen using DSS. This new hierarchy shall be referred to as the DSS-DSS hierarchy and is shown to improve the error properties of the ensuing solution.

The remainder of the paper consists of the methodology for hierarchical DSS, the ensuing Results and Discussion, Sections 2, 3 and 4 respectively.

2 Methodology

As indicated above, our principle interest lies in the investigation of an exemplar subsampling algorithm, which filters the dataset in proportion to the ‘age’ and ‘difficulty’ of exemplars as viewed by the learning algorithm, whilst also incorporating the concept of a memory hierarchy. Such a scheme should significantly decrease the time to complete the inner loop of GP, without impacting on the error performance. The algorithm is actually independent of the supervised learning algorithm, but in this case is motivated by the plight of GP in which the inner loop is iterated over a population of candidate solutions. Section 2.1 summarizes the form of GP utilized later (any generic form of GP will suffice), whereas the methodology for hierarchical dynamic subset selection is detailed in Sections 2.2 and 2.3.

2.1 Genetic Programming

In the case of this work a form of Linearly-structured GP (L-GP) is employed [9-12]. That is to say, rather than expressing individuals using the tree like structure popularized by the work of Koza [13], individuals are expressed as a linear list of instructions [9]. Execution of an individual therefore mimics the process of program execution normally associated with a simple register machine. That is, instructions are defined in terms of an opcode and operand (synonymous with function and terminal sets respectively) that modify the contents of internal registers {R[0],…,R[*k*]}, memory and program counter [9]. Output of the program is taken from the best register upon completion of program execution (or some appropriate halting criterion [11]), where the best register is the register of the best performing individual that generates the greatest number of hits. Moreover, in an attempt to make the action of the crossover operator less destructive, the Page-based formulation of L-GP is employed [12]. In this case, an individual is described in terms of a number of *pages*, where each page has the *same* number of *instructions*. Crossover is limited to the exchange of *single* pages between two parents, and appears to result in concise solutions across a range of benchmark regression and classification problems. Moreover, a mechanism for dynamically changing page size was introduced, thus avoiding problems associated with the *a priori* selection of a specific number of instructions per page at initialization. Mutation operators take two forms. In the first case the 'mutation' operator selects an instruction for modification with uniform probability and performs an Ex-OR with a second instruction, also created with uniform probability. If the ensuing instruction represents a legal instruction the new instruction is accepted, otherwise the process is repeated. The second mutation operator 'swap' is designed to provide sequence modification. To do so, two instructions are selected within the same individual with uniform probability and their positions exchanged.

2.2 RSS-DSS Hierarchy

The basic formulation for the hierarchical sampling of training exemplars divides the problem into three levels. Level 0 divides the training set into a sequence of equal blocks. Blocks reside in memory and are chosen stochastically, Level 1. Level 2 samples the exemplars of the selected block using a stochastic sampling algorithm biased towards the more difficult or older patterns, or Dynamic Subset Selection (DSS) [7]. Program 1 outlines the general relationship between learning algorithm (GP in this case) and hierarchical sampling algorithm for the case of RSS block selection at level 1 and DSS exemplar selection at level 2:

```
Program 1 - Genetic Programming with RSS-DSS Hierarchy
{
(1)  divide dataset into blocks (level 0)
(2)  initialize training system and population
(3)  while (RSStermination == FALSE)
     {
(4)    conduct Block Selection (level 1)
(5)    while (DSStermination == FALSE)
       {
(6)      conduct Subset Selection (level 2)
(7)      while (TournamentEnd == FALSE)
```

```
          {
(8)         conduct tournament selection
(9)         train tournament individuals on Subset
(10)        update connection difficulty
(11)        apply genetic operators
          }
        }
(12)    update #Subset selected at next block b instance
      }
(13) run best individual on entire dataset
(14) run best individual on test dataset
(15) record results
(16) remove introns and translate
}
```

Basic design decisions now need to identify: how a block is identified (level 1), how a subset is selected (level 2) where GP individuals only iterate over the contents of a subset, and how many subsets are selected per block, i.e. the source of the computational speedup. Two basic algorithms are proposed, RSS-DSS (following) and DSS-DSS (§2.3).

Level 1 – Block based Random Subset Selection (RSS). At level 0 the datasets are partitioned into 'blocks' (Program 1-1), all the blocks exist on the hard disk. A block is then randomly selected with uniform probability (Program 1-4) – or Random Subset Selection (RSS) – and read into RAM, level 1 of the hierarchical Subset Selection algorithm. Following selection of block '*b*' a history of training pressure on the block is used to determine the number of iterations performed at level 2 of the RSS-DSS hierarchy – the DSS subset. This is defined in proportion to the error rate over the previous instance of block '*b*' for the 'best' individual over a level 2 subset from block '*b*' (Program 1-12), $E_b(i\text{-}1)$. Thus, the number of DSS subset iterations, *I*, on block, *b*, at the current instance, *i*, is

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

Where $I_{(max)}$ is the maximum number of subsets that can be selected on a block; and $E_b(i - 1)$ is the error rate (number of block misclassifications) of the best-case subset individual from the previous instance, *i*, of block *b*. Hence, $E_b(i)=1 - [hits_b(i)/ \#patterns(b)]$, where $hits_b(i)$ is the hit count over block *b* for the best-case individual identified over the last level 2 tournament at iteration *i* of block *b*; and *#patterns*(*b*) is the total number of feature vectors in block *b*. Naturally denoting the 'best case' individual relative to those which took part in level 2 competitions has the potential to miss better performing individuals which might reside in the population. However, this would also reintroduce a significant computational cost of the inner loop. Specifically, an array is kept to record hits for each individual in the population. After each tournament the hits of the parents are accumulated while the hits of the newly created individuals are set to zero. The best case individual over block *b* is therefore the individual with the maximum number of hits accumulated.

Level 2 – Dynamic Subset Selection (DSS). A simplified version of DSS is employed in the second level of the selection hierarchy [7]. Once a block has been selected using the above RSS process, patterns within the block are associated with an age and difficulty. The age is the number of DSS selections since the pattern last appeared in a DSS subset. The difficulty is the number of GP individuals that misclassified the pattern the last time it appeared in the DSS subset. Following selection of a block at level 1, all ages are set to one and all difficulties are set to a worst-case difficulty (i.e. no persistence of pattern difficulties or ages beyond a block selection). Patterns appear in the DSS subset stochastically, with a fixed chance of being selected by age or difficulty (%difficulty = 100 – %age). Thus two roulette wheels exist per block, one is used to control the selection of patterns with respect to age and the other difficulty, the roulette wheels being selected in proportion to the corresponding probability for age and difficulty. This process is repeated until the DSS subset is full (Program 1-6), the age and difficulty of selected patterns being reset to the initial values. Patterns that were not selected from the block have their age incremented by one whereas their difficulties remain the same. Currently DSS uses a subset size of 50 patterns, with the objective of reducing the number of computations associated with a particular fitness evaluation. Each DSS subset is kept for six steady state tournaments (4 individuals taking part per tournament) before reselection takes place with up to $I_{(b)}(i)$ selections per block (Program 1-5) – equation (1).

The use of the fixed probability of 70% for difficulty ensures that greater emphasis is given to connections that resist classification, while the 30% for age ensures that easier patterns are also visited in an attempt to prevent over-learning.

2.3 DSS-DSS Hierarchy

The RSS-DSS algorithm makes the implicit assumption that all blocks are equally difficult. The following DSS-DSS algorithm relaxes this assumption. To do so, a block difficulty and age is introduced and used to bias the stochastic selection of blocks in proportion to their relative age and difficulty using roulette wheel selection. Thus, the probability of selecting block *i* is,

$$\text{Block}(i)_{\text{weight}} = \frac{\%\text{diff} \times \text{Block}_{\text{diff}}(i)}{\sum_j (\text{Block}_{\text{diff}}(j))} + \frac{\%\text{age} \times \text{Block}_{\text{age}}(i)}{\sum_j (\text{Block}_{\text{age}}(j))}$$

$$P(\text{block}(i)) = \frac{\text{Block}(i)_{\text{weight}}}{\sum_j (\text{Block}(j)_{\text{weight}})} \qquad (2)$$

Where %diff is the fixed difficulty weighting (70%) and %age the age weighting (100 - %diff); $\text{Block}_{\text{diff}}(i)$ and $\text{Block}_{\text{age}}(i)$ are the respective block difficulty and age for block *i*; and *j* indexes all blocks.

At initialization each block has an age of one and worst-case difficulty. Therefore, block selection will initially be uniform. The age of a block is the number of RSS block selections since last selected. The difficulty of a block takes the form of a weighted running average, thus persistent across block selections, or,

$$\text{Block}(i, 0) = \text{Block}\ (i - 1);$$
$$\text{Block}(i, j) = \alpha\ \textit{pattern}_{\text{diff}}(j) + (1 - \alpha)\ \text{Block}\ (i, j - 1);\ \forall j \in \{1, \ldots, P_{\text{subset}}\}$$
$$\text{Block}(i) = \text{Block}(i, P_{\text{subset}})$$

Where $\textit{pattern}_{\text{diff}}(j)$ is the difficulty of pattern *j*; j indexes all patterns in the subset of P_{subset} patterns; and, α is a constant ($0 < \alpha < 1$), set to 0.1 in this case. The difficulty of a block will be updated before each new level 2-subset selection (step Program 1-11 and before returning to Program 1-5). Since α is small, each of the patterns in subset have the ability to influence the overall block difficulty by a small amount. If the connection difficulty is high, the overall block difficulty will increase, whereas if the connection difficulty is small, the overall block difficulty will decrease. Level 2 of this hierarchy uses the same DSS process as the RSS-DSS hierarchy.

3 Results

As indicated in the introduction, the principle interest of this work is in establishing a framework for applying Genetic Programming to large datasets. To this end experiments are reported using two large datasets: the KDD-99 Intrusion Detection dataset, taken from the 5th ACM SIGKDD Knowledge Discovery and Data Mining Competition (1999) [14]; and the Adult dataset, taken from the UCI Machine Learning Repository [15]. The total number of patterns in the training and test sets of each dataset is listed in Table 1.

Table 1. DataSets: Sizes and Distributions

	KDD-99		Adult	
Connection	Training (10%KDD)	Test (Corrected Test)	Training	Test
Class 0	97,249	60,577	7,508	3,700
Class 1	396,744	250,424	22,654	11,360
Total	493,993	311,001	30,162	15,060

For KDD-99 two partitions are used, 10% KDD for training and Corrected Test for test, as per the original competition [14]. Each pattern is described in terms of 41 features, comprising of 9 basic features and 32 derived features describing temporal and content information. Here we only use the first 8 basic features, but express these in terms of a shift register with 8 taps taken at intervals of 8 sequential patterns. Such a scheme requires that the learning algorithm also identify the useful temporal properties, rather than relying on the *a priori* selected features (see [16]). Each entry of the

KDD dataset represents a connection, labeled in terms of one of five categories: Normal, Denial of Service (DoS), Probe, User to Root (U2R) and Remote to Local (R2L). In this work we are only interested in distinguishing between Normal and any of the four attack categories. Moreover, 79% of the training data represent instances of DoS, 20% normal and the remainder Probe, U2R and R2L, Table 2. Thus, as well as representing a large training set it is unbalanced, introducing the possibility for degenerate solutions i.e. a detector that labels every pattern as attack. The test data on the other hand increases the contribution of the smallest attack category, R2L, to 5% of the dataset and introduces 14 attack types unseen during training, Table 2.

Table 2. 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%

The Adult dataset is significantly smaller than KDD (30 thousand as opposed to half a million patterns), Table 1, but is introduced to verify that the proposed algorithms are not making undue use of duplicate properties that might exist in KDD. Specifically, the DoS class making up nearly 80% of the training data might well demonstrate self-similar properties (a natural characteristic of a DoS attack). Thus, resulting in the sampling algorithm ignoring these exemplars with little or no penalty once some classification capability on DoS has been identified. The adult dataset represents a prediction problem taken from the 1994 Census database. The learning task is to predict whether a person's income exceeds $50,000 per year based on 14 census data features. Each pattern represents a person and is made up of 14 personal and demographic characteristics plus a label. All patterns with missing features were removed from both the training and test data. The data distribution for the Adult dataset is approximately 25% class 0 and 75% class 1 for both training and test sets.

All the following experiments are based on 40 GP runs using Dynamic page-based Linear-GP [12]. Runs differ in their choice of random seeds used for initializing the population, all other parameters remaining unchanged. Table 3 lists the common parameter settings for all runs.

In addition to validating the use of the proposed hierarchical process for sampling patterns, experiments with different block sizes were made under the Adult dataset, Table 4. Note for each dataset the final block does not contain the full block size but the remaining number of patterns.

Instruction Set. The GP instructions employ a 2-address format 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) [12]. 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 register merely increases the size of the associated instruction field.

Table 3. Parameter Settings for Dynamic Page-based Linear GP

Parameter	Setting
Population size	125
Maximum # of pages	32
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} ∪ {pattern features}
Level 2 subset size	50
RSS iterations	1000
Max DSS iterations (6 tourn./iteration)	100
Wrapper function	0 if output ≤ 0, otherwise 1
Cost function	Increment by 1/(# in class) for each misclassification

Table 4. Dataset block sizes and number of blocks

Dataset	10% KDD-99	Adult	
Block size	5000	1000	5000
# of blocks	99	31	7

Training was performed on a dual G4 1.33 GHz Mac Server with 2 GB RAM. In all 6 experiments were run. Two experiments on the 10% KDD-99 dataset: the first using a hierarchy of RSS-DSS; and the second using the hierarchy of DSS-DSS. Four experiments were conducted on the Adult Dataset using the two hierarchies as well as the two different block sizes (1000 and 5000).

For each GP run in an experiment the best individual is recorded and simplified by removal of structural introns [17]. The 'best' individual is identified post training by taking the maximum of the hits accumulated by each individual on the entire training dataset. The performance of each best individual is expressed in terms of time, program length (before and after simplification), detection rate (DR) and false positive rates (FPR) on Corrected KDD Test set. Detection rates and false positive rates are estimated as follows,

$$\text{Detection Rate} = 1 - (\text{ \# of False Negatives / Total \# of Attacks }) . \quad (5)$$

$$\text{False Positive Rate} = (\text{ \# of False Positives / Total \# of Normals }) . \quad (6)$$

3.2 10% KDD-99 Dataset

Results are initially summarized in terms of first, second (median) and third quartiles for time, detection rate, false positive rate, and solution complexity before and after intron removal. In addition comparisons are made against reported figures for both training characteristics and quality of the resulting solutions.

Figure 1 indicates that in general the RSS-DSS hierarchy typically takes less time to train than the DSS-DSS hierarchy on the 10% KDD-99 dataset. This appears to be a natural reflection of the overhead in conducting an additional roulette wheel based calculation for block selection in DSS-DSS, as opposed to the uniform selection in the RSS-DSS algorithm. However, both algorithms are two orders of magnitude better than previously reported research in which GP was applied directly to the entire 10%KDD dataset [18]. Such a direct application required 48 hours to complete a single trial (Pentium III, 800MHz), whereas here each trial takes less than 15 minutes; 40 trials therefore completing in 10 hours.

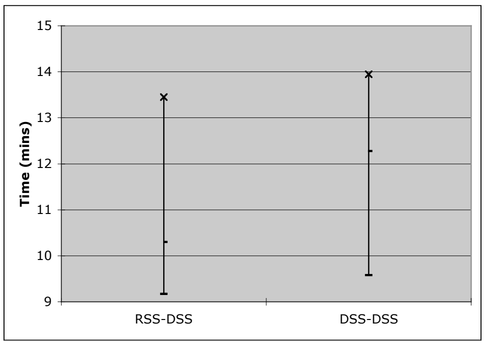

Fig. 1. KDD'99 – Training Time (first, second (median) and third quartile)

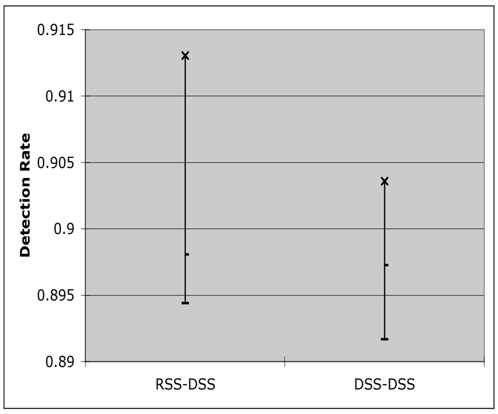

Fig. 2. KDD'99 – Detection Rate (first, second (median) and third quartile)

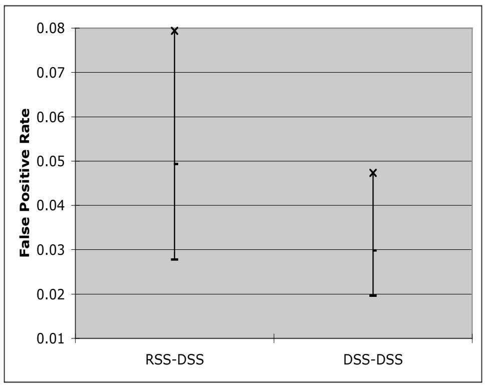

Fig. 3. KDD'99 – False Positive Rates (first, second (median) and third quartile)

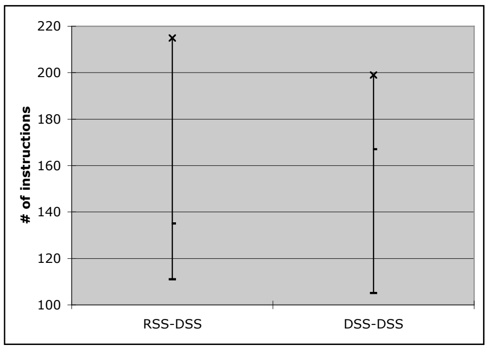

Fig. 4. KDD'99 – Solution Complexity (first, second (median) and third quartile)

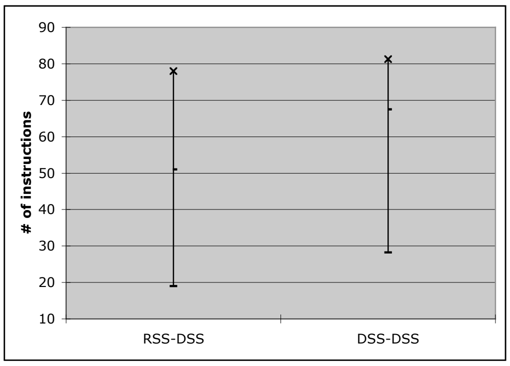

Fig. 5. KDD'99 – Solution Complexity (first, second (median) and third quartile)

Detection and False Positive (FP) Rates of the 'best' individual across the 40 runs, figures 2 and 3 respectively, indicates that uniform selection of blocks results in a significantly wider spread in both Detection and FP rates relative to that from DSS-DSS.

Moreover, the median FP rate of DSS-DSS is better than that for RSS-DSS. This property also appears to have carried over to the complexity of solutions, figures 4 and 5, with DSS-DSS returning longer solutions (i.e. more instructions) both before and after simplification. Moreover, the trend continues with respect to classification count of each category of attack, Figures 6 - 9. It is also apparent that DSS-DSS has emphasized solutions to the DoS category, where this represents 80% of the training/ test data.

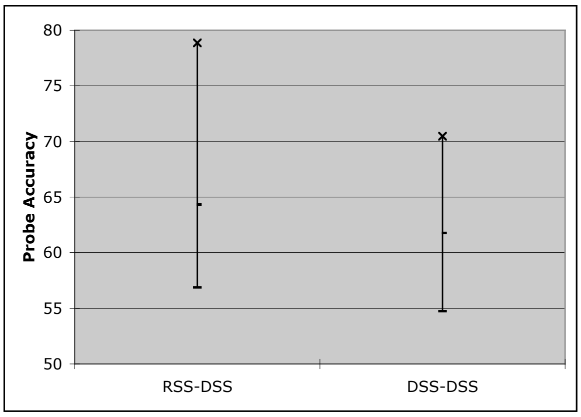

Fig. 6. Probe Accuracy.

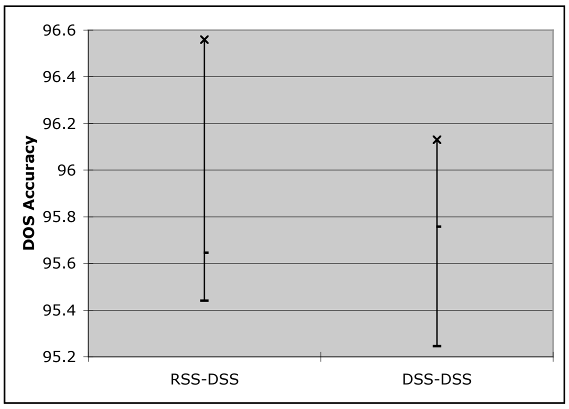

Fig. 7. Denial Of Service Accuracy

In order to qualify whether the hierarchical subset selection algorithm negatively impacts classifier quality, comparison is made against the results from the original KDD-99 competition winners, Table 2. Both the winning entries were based on decision trees, also taking roughly 24 hours to complete training. The resulting solutions were complex (500 C5 decision trees in the case of the winning entry) and trained using a boosting algorithm on different partitions of the original 10%KDD dataset. Moreover, independent detectors were developed for each class over all 41 features, making a direct comparison difficult. It is apparent however, that the GP solutions are competitive.

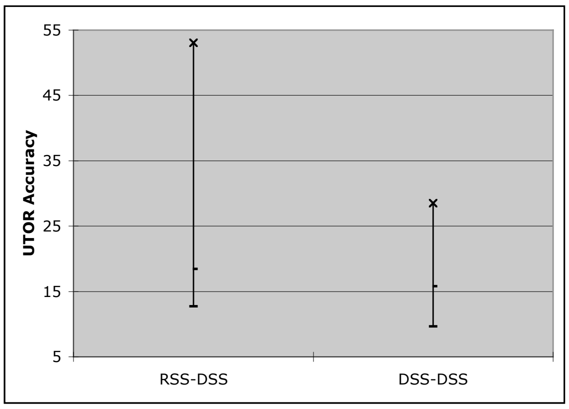

Fig. 8. User to Root (U2R) Accuracy.

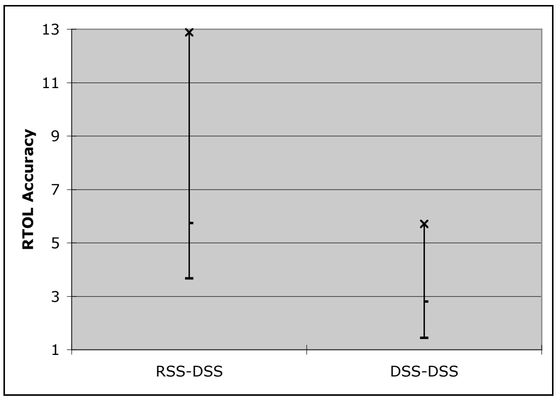

Fig. 9. Remote to Local (R2L) Accuracy

Table 5. Comparison with KDD-99 winning entries

Parameter	Detection Rate	FP Rate
Winning Entry	0.908819	0.004472
Second Place	0.915252	0.00576
Best GP (RSS-DSS)	0.889609	0.0128108
Best GP (DSS-DSS)	0.897184	0.0062568

3.3 Adult Dataset Results

In the case of the Adult dataset we also consider the significance of a smaller block size, thus all properties are expressed in terms of block sizes of 1000 and 5000. From Table 6 it is evident that a smaller block size results in a significant speedup in CPU time to complete a run. Such an observation mirrors that in cache block size design, with larger blocks encountering a higher transfer penalty if this is not leveraged into extended utilization. Relative to the KDD case, an extra five minutes appear to be necessary indicating that although the dataset is smaller, there is more diversity in the dataset, where this is also reflected in the additional complexity in the solutions, Table 6. Detection and FP rates, Table 6, demonstrate variation between RSS-DSS for different block sizes, whereas DSS-DSS retain consistency across the two block sizes. Moreover, median Detection rates for DSS-DSS exceed that for RSS-DSS and pair-wise comparison of median FP rates also prefers the DSS-DSS algorithm. In terms of solution complexity, there is no significant difference between the two algorithms, but a smaller block size appears to result in less complex solutions, Table 6. Thus, the optimal parameterization appears to be in the smaller block size – faster training time and simpler solutions – with the DSS-DSS algorithm.

Table 6. Performance of RSS-DSS and DSS-DSS on Adult Dataset

Run Time				
Algorithm	RSS-DSS		DSS-DSS	
Block Size	1,000	5,000	1,000	5,000
Median	10.32	18.46	11.53	15.97
Detection Rate				
1st Quartile	0.8129	0.8012	0.8254	0.8061
Median	0.8446	0.8253	0.8575	0.8537
3rd Quartile	0.9237	0.8471	0.9122	0.9013
False Positive Rate				
1st Quartile	0.2600	0.2302	0.2962	0.2597
Median	0.3300	0.2857	0.3549	0.3185
3rd Quartile	0.5403	0.3643	0.4630	0.4489
Solution Complexity – Before Intron Removal				
Median	127	179	143	171
Solution Complexity – After Intron Removal				
Median	62	88	60	83.5

In order to provide some qualification of the classifier performance Table 7 details (best case) error rates of alternative machine learning algorithms summarized as part of the UCI repository [15]. Unfortunately no information is provided regarding the learning algorithms or any preprocessing performed to achieve these results. Table 8 lists the error rates of the best-case RSS-DSS / DSS-DSS algorithms with block sizes of 5000, 1000, 500 and 250. It is readily apparent that the GP solutions are ranked towards the end of the list, however, they also appear before the step change in errors from an error of 17 to an error of 19.5. Moreover, no attempt was made to optimize parameters such as the ratio between difficulty and age (boosting algorithms in wide spread use are based on difficulty alone). We also note that as block size decreases, error rates also decrease whilst the preference for the DSS-DSS algorithm becomes more apparent.

Table 7. Error Rates on Adult Dataset

Algorithm	Error	Algorithm	Error
FSS Naïve Bayes	14.05	Voted ID3 (0.6)	15.64
NBTree	14.10	CN2	16.00
C4.5-auto	14.46	Naïve-Bayes	16.12
IDTM (decision table)	14.46	Voted ID3 (0.8)	16.47
HOODG	14.82	T2	16.84
C4.5 rules	14.94	1R	19.54
OC1	15.04	Nearest-neighbor (3)	20.35
C4.5	15.54	Nearest-neighbor (1)	21.42

Table 8. RSS-DSS and DSS-DSS Error Rates on Adult Dataset

Block Size	5000	1000	500	250
RSS-DSS	17.56	16.78	16.95	16.95
DSS-DSS	17.07	16.95	16.63	16.46

4 Conclusion

The computational overhead of the GP inner loop is addressed by introducing a hierarchy of training subset selections. This enables the scaling up of the size of problems for which approaches based on GP may be applied. To do so, the original problem is divided into a series of blocks. Blocks are either selected uniformly (RSS) or relative to their age and difficulty (DSS). Exemplars are sampled from a block relative to their relative block age and difficulty (DSS). Such a scheme matches the observations used to formulate the memory hierarchy typically employed in computer architectures. The ensuing algorithm, DSS-DSS, appears to be competitive with alternative learning algorithms previously applied, with the advantage that the computational overhead appears to be significantly reduced. The method differs from alterative sampling algorithms such as 'boosting' by introducing the concept of age and difficulty (boosting is explicitly based on exponentially weighted difficulties) and utilizing a hierarchy of samples. The latter point is fundamental in efficiently scaling the algorithm too much larger datasets than is the norm with GP. That is to say, competitive solutions are located in minutes rather than hours, and the framework is not specific to GP, thus potentially applicable to other learning algorithms such as neural networks.

Acknowledgements.

References

1. Bennett III F.H. et al.: Building a Parallel Computer System for $18,000 that Performs a Half Petra-Flop per Day. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Morgan Kaufmann (1999) 1484-1490

2. Juillé H., Pollack J.B.: Massively Parallel Genetic Programming. In: Angeline P.J., Kinnear K.E. (eds): Advances in Genetic Programming 2, Chapter 17. MIT Press, Cambridge, MA (1996) 339-358
3. Koza J.R. et al.: evolving Computer Programs using Reconfigurable Gate Arrays and Genetic Programming. Proceedings of the ACM 6th International Symposium on Field Programmable Gate Arrays. ACM Press. (1998) 209-219
4. Breiman L.: Bagging predictors. Machine Learning. 24(2) (1996) 123-140
5. Freund Y., Schapire R.E.: A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. Journal of computer and Systems Sciences. 55 Academic Press. (1997) 119-139
6. Hennessy J.L., Patterson D.A.: Computer Architecture: A Quantitative Approach. 3rd Edition. Morgan Kaufmann, San Francisco, CA (2002)
7. 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
8. Song D., Heywood M.I., Zincir-Heywood A.N.: A Linear Genetic Programming Approach to Intrusion Detection. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO). Lecture Notes in Computer Science. Vol. 2724 Springer-Verlag. (2003) 2325-2336
9. 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
10. Nordin P.: A Compiling Genetic Programming System that Directly Manipulates the Machine Code. In: Kinnear K.E. (ed.): Advances in Genetic Programming, Chapter 14. MIT Press, Cambridge, MA (1994) 311-334
11. Huelsbergen L.: Finding General Solutions to the Parity Problem by Evolving Machine-Language Representations. Proceedings of the 3rd Conference on Genetic Programming. Morgan Kaufmann, San Francisco, CA (1998) 158-166
12. 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
13. Koza J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA (1992)
14. Elkan C.: Results of the KDD'99 Classifier Learning Contest. SIGKDD Explorations. ACM SIGKDD. 1(2), (2000) 63-64
15. UCI Machine Learning Repository. (2003) http://www.ics.uci.edu/~mlearn/MLRepository.html
16. Lichodzijewski P., Zincir-Heywood A.N., Heywood M.I.: Host-Based Intrusion Detection Using Self-Organizing Maps. IEEE-INNS International Joint Conference on Neural Networks. (2002) 1714-1719
17. Brameier M., Banzhaf W.: A Comparison of Linear genetic Programming and Neural Networks in Medical data Mining. IEEE Transaction on Evolutionary Computation. 5(1) (2001) 17-26
18. Chittur A.: Model Generation for Intrusion Detection System using Genetic Algorithms. http://www1.cs.columbia.edu/ids/publications/ (2001) 17 pages