Scaling Genetic Programming to Large Datasets using Hierarchical Dynamic Subset Selection*

Robert Curry, Peter Lichodzijewski, and Malcolm I. Heywood†

July 26, 2007

Abstract

The computational overhead of Genetic Programming (GP) may be directly addressed without recourse to hardware solutions using active learning algorithms based on the Random or Dynamic Subset Selection heuristics (RSS or DSS). This work begins by presenting a family of hierarchical DSS algorithms: RSS-DSS, cascaded RSS-DSS, and the Balanced Block DSS algorithm; where the latter has not been previously introduced. Extensive benchmarking over four unbalanced real-world binary classification problems with 30,000 to 500,000 training exemplars demonstrates that both the cascade and Balanced Block algorithms are able to reduce the likelihood of degenerates, whilst providing a significant improvement in classification accuracy relative to the original RSS-DSS algorithm. Moreover, comparison with GP trained without an active learning algorithm indicates that classification performance is not compromised, while training is completed in minutes as opposed to half a day.

*Paper published in IEEE Transactions on Systems, Man, and Cybernetics: Part B. 37(4) 2007

†Faculty of Computer Science, Dalhousie University, NS. Canada

1 Introduction

Genetic Programming (GP) is a supervised machine learning paradigm in which multiple candidate solutions are investigated concurrently (the population) [1]. Such an approach provides the practitioner with unique opportunities for resolving the principle machine learning questions of cost (fitness) function design, credit assignment (exploration-exploitation tradeoff), and representation. However, such flexibility comes at a significant computational cost, typically limiting the application of the approach to relatively small datasets; that is, at most thousands of training exemplars. This is basically a factor of the cost associated with evaluating the performance of a population of individuals across the training dataset and the stochastic nature of the learning algorithm i.e., multiple runs are required over different initial populations in order to establish the statistical significance of results. Thus, typical parameters for a canonical GP [1] might imply 50 iterations of a population of 4,000 individuals over at least 30 runs, or a total of six million fitness evaluations; where each evaluation requires that the error be evaluated across the entire training dataset. Each time a parameter is modified the entire process is repeated.

In this work we are interested reducing the significance of the computational overhead in evaluating fitness without recourse to hardware speedups. The basic model investigated in this work is that of active learning. Such a paradigm recognizes that as the model(s) produced by the learning algorithm improve at each iteration, why not use the performance of the models to guide the selection of exemplars used for training. Thus, as model performance improves the number of exemplars over which training is conducted also decreases. Within the context of GP the active learning paradigm most widely used is that of Random Subset Selection (RSS) and Dynamic Subset Selection (DSS) [2]. The DSS framework introduces the concepts of exemplar age and difficulty to bias

the stochastic selection of training exemplars appearing in the subset over which GP individuals are actually trained. Recently, the concept of hierarchical DSS active learning algorithms for GP was introduced in order to scale the DSS approach to very large datasets (i.e., those that exceed cache memory) [3, 4, 5].

One disadvantage of the hierarchical DSS approach to active learning is that relationships between minor and major class distributions are fixed. This can lead to degenerate solutions in which all exemplars are labeled as the major class. In this work a hierarchical model for constructing training subsets is proposed for which the distribution of class exemplars is no longer fixed, the Balanced Block DSS algorithm. In addition, the concept of a cascaded model building is investigated, where the computational cost of such a scheme has previously precluded the use of such a model under a GP context. Extensive benchmarking over large unbalanced real-world binary classification datasets (30,000 to 500,000 training exemplars) demonstrate the superiority of the Balanced Block and cascaded schemes at avoiding degenerate solutions. Ultimately, the cascaded model provides the highest accuracy, whereas the Balanced Block algorithm provides better accuracies than the original hierarchical active learning scheme whilst maintaining the same computational cost.

In the following, Section II provides a short survey of related work. Section III establishes pertinent background to GP supervised learning, and summarizes the previous and proposed DSS based active learning algorithms. Details of the original hierarchical RSS-DSS active learning algorithm [3, 4, 5] and the Balanced-Block DSS algorithm proposed by this work are given in Section IV. In addition a Cascaded GP methodology [6, 7] is also summarized, where this requires the hierarchical RSS-DSS scheme to maintain computational feasibility. The experimental methodology is established in Section V, and performance detailed in Section VI. Section VII concludes the paper.

2 Related Work

2.1 Ensemble Methods

Within the context of supervised learning, the significance of data sampling algorithms have been widely acknowledged albeit with the motivation to improve error performance. Bagging and boosting are both data sampling algorithms that have the capacity to improve weak learners in that they exploit the instability inherent in the learning algorithms. Although GP is not a weak learner, favorable results have been reported using ensemble methods. Specifically, Iba and then Paris *et al.* applied boosting to GP with an emphasis on error minimization, and indeed found that boosting greatly improved GP performance [8, 9]. Their focus did not concern large datasets and indeed the use of large datasets would not be feasible under the scheme proposed. Furthermore, both bagging and boosting, as originally defined, do not directly address the computational overhead of GP fitness evaluation or the overhead associated with dealing with datasets that do not fit within RAM alone. However, recent work has combined parallel implementaions of GP (i.e., explicit hardware support in the form of a Beowulf cluster) with ensemble learning. The technique requires that the training data be partitioned and spatially distributed, with different partitions of the dataset appearing at different nodes in the Beowulf cluster. Different classifiers are trained on different subsets of the dataset and recombined using a suitable voting scheme [10].

2.2 Limited Error Fitness

Using the limited error fitness (LEF) algorithm of [11] a GP individual's fitness is related to how many of the ordered set of training exemplars it classifies correctly before it makes a certain number of misclassifications. After exceeding

this error limit any cases not yet covered by the individual are counted as misclassified. The fitness score is then the total number of misclassified exemplars; thus, it is quicker to find the fitness of a poor GP individual than a good GP individual which saves CPU time. However, the LEF algorithm introduced several parameters for controlling the change in difficulty of the problem in response to the performance of the population in the previous generation [11]. LEF was sensitive to the choice of values for these parameters. Furthermore, even though individual fitness evaluations are quick, LEF was found to require many more generations, further impacting the computational overhead of the algorithm.

2.3 Topology-Based Subset Selection

The authors of [12] developed a subset selection method that gathers information about the problem structure being examined during the evolutionary search. Such information is expressed as a topology on the set of fitness cases, where the topology is represented by an undirected weighted graph. The authors maintain that this topology-based selection (TBS) helps to improve the performance of GP by allowing dynamically smaller and more suitable subsets to be selected. The most computationally expensive tasks in TBS are the adaptation of the topology and the requirement that the edge values in the graph need to be sorted to select the subset at each generation. This scales approximately quadratically with training set size. The sorting of the edge weights for a dataset of size N in just one generation would be on the order of $\mathcal{O}(N^2 \log N^2)$ [12].

3 Genetic Programming and Active Learning

GP is a supervised machine learning algorithm based on the neo-Darwinian concepts of natural selection and survival of the fittest [1]. Operators responsible for addressing the credit assignment problem therefore take the form of a popu-

1. Initialize population of Programs;
2. Read training dataset;
3. while (stop criteria NOT satisfied)
   (a) Select programs for fitness evaluation (members of the tournament);
   (b) while (program < TournamentSize)
      i. while (pattern < NumPatterns)
         A. Run program on pattern;
      ii. Update fitness of program;
   (c) Apply search operators;
   (d) Update population;

Figure 1: Generic Genetic Programming Training Algorithm

lation based selection operator and genetically motivated search operators. The algorithm is inherently iterative, with the population of candidate solutions first being ranked using their relative fitness i.e., performance on the training data (Figure 1). A selection operator stochastically selects individuals for reproduction, Step 3(a). Search operators are stochastically applied to build children appearing in the next population, Step 3(c). Such search operators take their motivation from genetics and typically take the form of mutation and crossover. Steps 3(a) to 3(d) constitute a training epoch or 'tournament' and repeat until the termination criterion is satisfied i.e., when a predefined performance target is reached or after a fixed number of tournaments has elapsed.

From Figure 1 it is apparent that the time required to build solutions is proportional to the number of tournaments and the size of the dataset. In fact, relatively little CPU time is expended on other tasks of the algorithm, such as the creation of the initial random population and the application of search operators. In this work it is proposed that there are at least two aspects to the problem of efficiently training GP on large datasets. Firstly, there is the widely acknowledged computational overhead associated with establishing pro-

gram fitness, or the GP inner loop (Figure 1 Step 3(b)). This overhead has traditionally been addressed by some form of hardware solution. Specific examples include evolution at the level of machine language [13] or by parallelizing fitness evaluation using systems such as Beowulf clusters [14, 10], super computers [15], and Field Programmable Gate Arrays [16]. Secondly, there is a less widely acknowledged overhead, that of managing datasets that cannot reside within RAM alone. For datasets of much more than a thousand or so records, reading the entire dataset into cache may not be possible. Therefore, sequentially presenting the entire dataset to each individual at each generation (Figure 1, 3(b)i) will not make efficient use of the localized spatial and temporal access patterns on which cache memory and memory hierarchies are based [17]. As mentioned in the introduction these problems will be addressed through recognizing 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.

In order to address these points we build on the Dynamic Subset Selection (DSS) active learning algorithm [2] and a hierarchical variant RSS-DSS [3, 4, 5]. This provides the motivation for the Balanced Block DSS algorithm introduced in this work. Moreover, in order to provide a complete picture of the various performance tradeoffs, the cascaded variant of the RSS-DSS algorithm [6, 7] is also benchmarked.

3.1 Hierarchical Subset Selection Algorithms

3.1.1 Hierarchical Random and Dynamic Subset Selection

For very large datasets that do not fit within RAM alone, the use of subset selection algorithms will still require multiple slow hard disk accesses in order to support the selection of subsets. Therefore, in the work of [3, 4] the

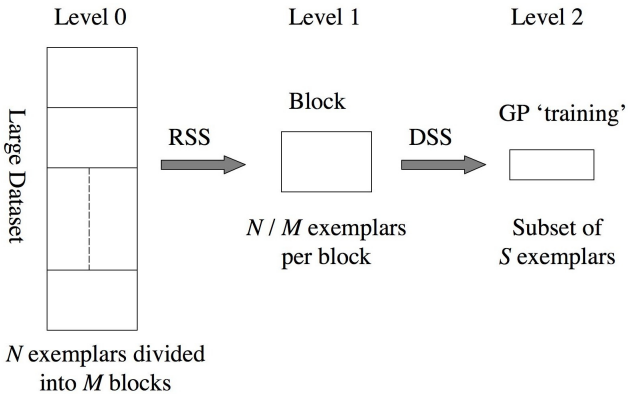

Figure 2: RSS-DSS Hierarchical Subset Selection Architecture.

Dynamic Subset Selection (DSS) algorithm was extended into a hierarchy of subset selections, to match the concept of memory hierarchies as supported in modern computers. This hierarchical subset selection algorithm has been re-implemented in this work for benchmarking purposes. First the entire training dataset is partitioned into blocks that are small enough to fit within RAM (Figure 2 level 0). Blocks are then chosen from the partitioned dataset with uniform probability, that is Random Subset Selection (RSS). This forms level 1 of the selection hierarchy. Level 2 of the selection hierarchy used the method of DSS to stochastically select exemplars form the block identified at level 1, biased by exemplar difficulty and age. Multiple level 2 subsets are selected for each level 1 block selection. Hereafter we refer to this as the RSS-DSS algorithm.

3.1.2 Cascaded GP

The availability of active learning algorithms, such as the RSS-DSS algorithm, provide the opportunity to build classifiers with a greater degree of accuracy than was previously the case. To do so, use was made of the cascade correlation architecture originally proposed within the context of neural networks [18, 19]. The approach provides the basis for incrementally building increasingly

sophisticated classifiers. Such a scheme provides the opportunity to refocus each additional classifier on the remaining residual error, with respect to models previously built. Naturally, classifiers comprising the cascade are trained over the entire training dataset, with the feature set being augmented with the results from each previous GP model output, resulting in an increase in the dimensionality of the original training dataset. Needless to say, such a scheme is only feasible for GP when utilizing an active learning algorithm, in this case RSS-DSS [6, 7].

3.1.3 Balanced Block Dynamic Subset Selection

A potential drawback of the original RSS-DSS algorithm was that no attempt was made to explicitly ensure that subsets over which GP classifiers are trained contain both in and out of class exemplars. As a consequence degenerate solutions might well dominate the population. To this end we introduce the Balanced Block Dynamic Subset Selection algorithm, or BB-DSS. The principle motivation for this scheme is to organize the content of each block such that the distribution of exemplar classes has a fixed ratio, where this need not be limited to the original class distribution. Indeed, in the case of very unbalanced datasets, it may be advantageous for the minor class to be over represented relative to the original distribution. To do so, the BB-DSS algorithm first divides the training dataset into its separate classes (Figure 3 level 0). The separate classes are then divided into 'partitions' corresponding to the distribution of that class for representation in the block. Level 1 block selections are then defined in terms of selecting a partition from each class by means of the DSS algorithm. This requires that the concepts of age and difficulty be extended to the partitions of each exemplar class. Partitions are now chosen stochastically, with a bias towards partition difficulty and age. This naturally relaxes the assumption made by the RSS-DSS hierarchy that all blocks are equally difficult [3, 4, 5].

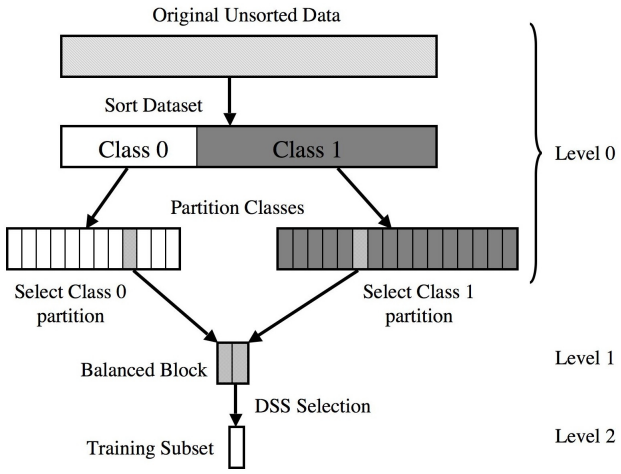

Figure 3: Balanced Block DSS Architecture

Detailed descriptions of all three hierarchical subset selection algorithms will follow in Section 4.

4 Algorithm Details

As indicated above, our principle interest lies in the investigation of exemplar sub-sampling algorithms, which filter the dataset in proportion to the 'age' and 'difficulty' of exemplars as viewed by the learning algorithm, while also incorporating the concept of a memory hierarchy. To this end, the methodologies for the three hierarchical subset selection techniques are detailed in Sections 4.2, 4.3 and 4.4. However, we begin by summarizing the specific type of GP utilized in this work (for a tutorial level presentation of GP see [21]), although any generic form of GP will suffice, Section 4.1.

4.1 Linearly-structured GP

In this work the GP representation takes the form of a linear structure, hereafter referred to as L-GP [13, 20, 21]. That is to say, rather than expressing individu-

als using the tree like structure popularized by the work of Koza [1], individuals are expressed as a linear list of instructions which are executed sequentially. Execution of an individual therefore mimics the process of program execution normally associated with a simple register machine. Instructions are defined in terms of an opcode and operand that modify the contents of general purpose registers {R[0],...,R[k]}, memory and program counter. Output of a program is taken from a predefined register upon completion of program execution.

The selection operator takes the form of a steady state tournament (Figure 1, Step 3(a)). This means that rather than attempting to build an entirely new population at each 'generation' using a roulette wheel and fitness proportional selection [1], we select a much smaller subset of individuals of size T (the size of the tournament). Only these individuals have their fitness evaluated, with the best $T/2$ individuals providing children through application of the search operators. The children replace the individuals in the original population corresponding to the worse performing $T/2$ individuals of the tournament. Such steady state schemes have been shown to have higher takeover rates than generational based selection operators and have been applied in both L-GP [13, 21, 22] and GA contexts [23].

In an attempt to make the action of the crossover operator less destructive, the location of crossover points remains constant [22]. 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. 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 replaces

it with a different instruction from the instruction set. 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. The motivation behind the swap operator is that the sequence in which instructions are executed within a program has a significant effect on the solution. Thus, a program may have the correct composition of instructions but specified in the wrong order.

The above constraints on the operation of the crossover operator implies that once initialized the length of an individual (the number of pages multiplied by the number of instructions per page) never changes. As a consequence the population is initialized over the permitted range of program lengths ([8, ..., 256] in this work).

4.2 RSS-DSS Algorithm

The basic formulation for the hierarchical sampling of training exemplars divides the problem into three levels (Figure 2). Level 0 divides the training set into a sequence of equal blocks. Blocks reside in memory and are chosen stochastically with uniform probability or RSS (Figure 2). Dynamic Subset Selection (DSS) samples the exemplars of the selected block stochastically, with a bias towards selecting the more difficult or older exemplars [2, 3, 4, 5]. Figure 4 outlines the general relationship between GP learning algorithm, Steps 2 and 3(b).ii to 3(b).v (highlighted in italic), and the generic form of the hierarchical active learning algorithm.

In the case of the RSS-DSS algorithm, a block of exemplars is selected with uniform probability (RSS) at level 1, Step 3(a). This set of exemplars is then sub-sampled by way of DSS at level 2, Step 3(b).i, to form the set of exemplars over which fitness evaluation is actually performed. Updating of the DSS

parameters takes place at Step 3(b).iii (exemplar difficulty) and 3(b).vi (exemplar age). The process is then iterated for different tournament individuals and subset composition. On reaching the stop criterion for DSS sub-samples taken from the current block, the block error is updated, Step 3(c). Specifically, as the 'block' error decreases the number of DSS iterations per block decreases, resulting in a corresponding computational speedup [2, 3, 4, 5]. Block error is approximated by evaluating the error of the best performing individual from DSS, Step 3(b), but over all exemplars in the current block.

It is apparent that blocks are selected with uniform probability, level 1, whereas subsets are selected at level 2 with a bias towards the more difficult or least frequently selected (block) exemplars. Implicit in this model is the assumption that all blocks are equally difficult, and that a sufficient distribution of minor (major) class exemplars exists per block during training. Moreover, the algorithm also assumes that the distribution of class exemplars per block is fixed. These assumptions are a natural consequence of the block based model used to support an efficient memory access model.

4.3 Cascaded GP Algorithm

The cascade architecture, hereafter CasGP [6, 7], adopts the scheme popularized by the cascade correlation neural network architecture [18] and further investigated by Litmann and Ritter [19]. Figure 5, summarizes the basic process. Models are built incrementally with the output from the best individual at each model augmenting the dataset (Figure 5, Step 1(b)). Thus, model 'm' receives an exemplar composed of the original features plus the output from the 'm – 1' previous models on the same exemplar (i.e., the final solution consists of a 'cascade' of up to 'm' independent GP models). Naturally, an active learning algorithm is necessary to ensure the computational feasibility of the approach

1. Stratify training data (**level 0**);
2. *Initialize population of Programs*;
3. *While* (*stop criterion == FALSE*)
   (a) Select Block (**level 1**);
   (b) While (DSStermination == FALSE)
      i. IF (DSSiteration MOD (SubsetFrequency))
         A. THEN (Identify training subset using DSS (**level 2**));
      ii. *Select programs for fitness evaluation* (*members of the tournament*);
      iii. *while* (*program < TournamentSize*)
         A. *while* (*pattern < NumSubsetPatterns*)
            Run program on pattern;
            Update error;
            Update exemplar difficulty;
      iv. *Rank programs*;
      v. *Apply search operators*;
      vi. Update exemplar age;
   (c) Update block error;
      i. DSStermination = *f(*MaxDSSiteration, BlockError);

Figure 4: Generic Hierarchical DSS Algorithm. Standard GP training algorithm steps in italic.

1. for (model = 0; model < maxModel; model++)
   (a) GP(model) $\leftarrow Train_{RSS-DSS}$(data);
   (b) cat(data, GP(model).output);
   (c) IF (minERR > GP(model).SSE) THEN
      i. minERR = GP(model).SSE;
      ii. bestModel = model;

Figure 5: Cascade Algorithm for incrementally building GP models.

(Figure 5 Step 1(a)). The RSS-DSS algorithm is utilized for this purpose, in line with the original CasGP algorithm [6, 7]. In addition, a record is taken of which model (i.e., layer) minimizes the training error. That is to say, given the stochastic nature of the GP paradigm there is no guarantee of monotonic improvements in performance across models. Test set performance is therefore evaluated using the model that performed best during training (Figure 5 Step 1(c)).

4.4 BB-DSS Algorithm

One approach to the problem of unevenly distributed data might be to stratify the data such that each level one block has the same distribution as the original data [5]. Unfortunately, when an exemplar class is very rare this scheme will also fail since rare exemplars are still encountered very infrequently. The approach taken here is to always have class 0 and class 1 exemplars represented in the Level 1 block at a fixed ratio. In this way both classes can always be represented regardless of the original size and distribution of the dataset. This approach is termed the Balanced Block DSS algorithm, or BB-DSS.

The first stage of the BB-DSS algorithm is to stratify the original dataset and to select the desired ratio of class 0 and class 1 exemplars to appear in the Level 1 balanced block. Once a ratio has been selected, say 25/75 (25%

of the block class 1 and 75% of the block class 0), the class 1 and class 0 exemplars in the stratified dataset can be divided up into partitions (Figure 3 level 0). For example, a block size of 1000 with a 25/75 block partition ratio would require class 1 partitions of 250 exemplars and class 0 partitions of 750 exemplars. A block can now be created by selecting a partition from each class and combining the two partitions (Figure 3 level 1). Moreover, the number of class 1 and 0 partitions need not be equal. The net effect is that every 'block' is constructed dynamically and consists of a set of exemplars at a predefined class ratio, independent of the initial data distribution.

4.4.1 Partition Selection

Partitions are selected for inclusion in the (level 1) block by extending the DSS algorithm, i.e., selection is biased through the concepts of partition difficulty and age. Specifically, partition age merely takes the form of a count, with the least frequently selected partition having a higher probability of selection. Partition difficulty is proportional to the difficulty of exemplars associated with the partition. Again the most difficult partitions will have a higher probability of selection.

In line with the above model, the probability of selecting partition i to represent its class in the level 1 block (Figure 4 Step 3(a)), takes the form:

$$Prob\left(Part(i)\right) = \frac{Part_{weight}(i)}{\sum_j Part_{weight}(j)} \tag{1}$$

$$Part_{weight}(i) = \frac{diff \times Part_{diff}(i)}{\sum_j Part_{diff}(j)} + \frac{age \times Part_{age}(i)}{\sum_j Part_{age}(j)}$$

where *diff* is the fixed percent difficulty weighting; *age* is the percent age weighting or, $age = (1 - diff)$; $Part_{diff}(i)$ and $Part_{age}(i)$ are the respective partition

difficulty and age for partition i; and j indexes all partitions in a class.

At initialization each partition has an age of one and a worst-case difficulty. Therefore partition selection will initially be uniform. The age of a partition is the number of partition selections for that class since the partition was last selected. The difficulty of a partition, $Part_{diff}(i)$, takes the form of an exponentially weighted running average, thus persistent across partition selections and robust to the performance on any one partition. Naturally, partition difficulty is initialized with the previous value, or

$$Part_{temp\_diff}(i, 0) = Part_{diff}(i - 1); \tag{2}$$

Once initialized, the partition difficulty receives an incremental update from each exemplar evaluated (Figure 4 Step 3(b).iii.A),

$$\begin{aligned} Part_{temp\_diff}(i, t) = \alpha\, exemplar_{diff}(t) \\ +(1 - \alpha)\, Part_{temp\_diff}(i, t - 1); \end{aligned} \tag{3}$$

When the last loop on the current block is performed, the partition difficulty is recorded, ready for initialization of the partition next time it appears in a block, or

$$Part_{diff}(i) = Part_{temp\_diff}(i, t) \tag{4}$$

where t indexes all exemplars in the current subset associated with partition i; $exemplar_{diff}(t)$ is the difficulty of exemplar t; and α is a constant $(0 < \alpha < 1)$

set to 0.1 in this case.

4.4.2 Subset Sampling Limit

It is now necessary to define the number of subset selections performed at level 2 of the hierarchy (Figure 4 Step 3(b)). To do so, use is made of the error on each partition last time they were evaluated. Let E_p denote the error rate of partition p, as defined by evaluating the program with best subset error over all exemplars in the block (Figure 4 Step 3(c)). The previous instance of a level 1 block in which the partition was a participant is indexed by i_p. The number of subset selections is now defined in proportion to the previous performance on each partition comprising the block. Thus the number of subset selections, *SubsetLimit*, on the current block is,

$$SubsetLimit = \frac{I_{max}}{2} \sum_{p \in Block} E_p(i_p - 1) \tag{5}$$

where I_{max} is the maximum number of subset selections allowed per block, and E_p is normalized to the unit interval, hence the factor of '2'. When both partitions comprising the block return maximum error (i.e., both unity) *SubsetLimit*$= I_{max}$. Defining the number of subset selections in this way allows GP to reduce the number of training rounds over partitions that were previously found to produce a low error rate and thereby reduces computational overhead.

4.4.3 Block Sampling Limit or Global Stopping Criterion

A method is now necessary for determining the number of level 1 block selections made (Figure 4 Step 3). Similar to the approach used for establishing updates to partition difficulty, an exponential weighted running average is again used,

$$AvgClassErr(i) = \beta\ ErrDiff + (1 - \beta)\, AvgClassErr \tag{6}$$

This error is updated at the end of a level 2 round (Figure 4 Step 3(c)), where β is a constant (set to 0.1), and *ErrDiff* is the absolute difference in error before and after training partitions comprising the level 1 block:

$$ErrDiff = |\triangle[E_p(i) - E_p(i-1)]| \tag{7}$$

The absolute difference in the Average Class Error, (6), after training is completed on a level 1 block provides a gradient, (8). This gradient is compared to a predefined threshold, ξ, to characterize whether convergence on the current block has taken place:

$$\begin{gathered} Gradient(i) = |\triangle AvgClassError|; \\ IF\ ((Gradient(i) < \xi)\ AND\ (Gradient(i-1) < \xi)) \\ THEN\ (BlockCount{+}{+})ELSE\ (BlockCount = 0); \end{gathered} \tag{8}$$

Moreover, by requiring that the block stop criterion is satisfied over consecutive level 1 block selections, in this case 15, we provide the overall stopping criterion for the Balanced Block Algorithm (Figure 4 Step 3). The corresponding block convergence threshold, ξ, is set to 0.0025, with index i denoting the count of consecutive block selections. Finally, given that there are two classes (i.e., partitions) comprising any block, we also require that the stopping criterion holds true for both classes.

Table 1: Characterization of Datasets

Dataset	Adult		Census	
Partition	Training	Test	Training	Test
Class 0	7,474	3,700	5,479	2,683
Class 1	22,688	11,360	89,651	44,708
Total	30,162	15,060	95,130	47,391
Dataset	Shuttle		KDD'99	
Class 0	9,392	3,022	97,249	60,577
Class 1	34,108	11,478	396,744	250,424
Total	43,500	14,500	493,993	311,001

5 Experimental Methodology

5.1 Experimental Setup

As indicated in the introduction, the principle interest of this work is to survey a family of DSS active learning algorithms used for applying GP to large datasets. To this end, experiments are reported using four large binary classification datasets: the KDD'99 Intrusion Detection dataset (the 10% KDD'99 dataset is used for training while the corrected KDD'99 dataset is used for testing), taken from the 5th ACM SIGKDD Knowledge Discovery and Data Mining Competition (1999) [24]; the Adult dataset, the Census Income dataset and the Shuttle dataset all taken from the UCI Machine Learning Repository [25, 26]. The distribution and sizes of the four datasets is shown in Table 1. The Shuttle dataset is originally a multi-class classification problem which was converted to a binary classification problem by labeling class 1 exemplars as negative and all remaining classes as positive exemplars. Moreover, in the case of the KDD'99 dataset, we are only interested in classifying normal behaviour (i.e., anomaly detection), thus also simplifying this dataset to a binary classification problem. The Census dataset is particularly unbalanced, with the minor class represented at 5.76%; whereas the most balanced dataset was Adult with 24.7% representing the minor class.

The stochastic basis adopted by the GP paradigm provides a mechanism for investigating different decisions given the same search state. Unlike deterministic machine learning algorithms it is then important to establish that solutions are not due to random chance; implying that runs be conducted over at least 30 different initializations [27]. Runs differ only in their choice of random seeds used for initializing the population, with all other parameters remaining unchanged. Moreover, given the significance of class imbalance, the number of degenerate solutions returned is recorded and performance metrics are reported over the remaining runs.

The output provided by each classifier takes the form of a real value over the interval (-1, 1). Such a scheme provides the basis for quantifying the degree of separation associated with classifier behavior as opposed to relying on a mere count of correctly classified exemplars. To do so, each classifier is based on a sigmoid wrapper function, mapping the raw GP output (GPout) to the required interval,

$$y = \frac{2}{(1 + \exp(-GPout))} - 1 \tag{9}$$

with fitness established using a sum square error fitness function, estimated over all 'P' training exemplars in the current subset,

$$J_{SSE} = \sum_{p \in P} \left(d(p) - y(p)\right)^2 \tag{10}$$

For the CasGP algorithm such a fitness function was previously observed to perform significantly better than a cross-section of alternative wrapper-fitness function pairs [6], as well as matching the performance of fitness functions explicitly designed to promote fitness sharing between cascade layers [7].

Training was performed on a dual G4 1.33 GHz Mac Server with 1 GB

RAM. The best individual program of a GP run was determined by running all programs in the population on the training dataset and selecting the program that minimized the training error rate. The performance of the best individual from each of the 30 GP runs was recorded in terms of training time in minutes, error rate on the training dataset and error rate, detection rate (DR) and false positive rate (FPR) on the test dataset. Detection rates and false positive rates are estimated as follows,

$$Detection\ Rate = 1 - \frac{\#\ False\ Negatives}{Total\ \#\ of\ Positives}$$

$$False\ Positive\ Rate = \frac{\#\ False\ Positives}{Total\ \#\ of\ Negatives}$$

5.2 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, 8 opcodes, and an 8-bit integer field representing constants (0-255) [22]. 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. Table 2 lists the common parameter settings for all GP runs.

5.3 BB-DSS Partition Parameterization

In order to compare the results of the RSS-DSS, CasGP and BB-DSS algorithms the parameterization of the BB-DSS algorithm in terms of the ratio of class 0 to class 1 exemplars in the level 1 block was investigated. T-tests estimated over the performance metrics for partition ratios of 20/80 and 37.5/62.5 on each of

Table 2: Parameter Settings for Dynamic Page-based Linear GP

Page Based Linear GP	
Parameter	Setting
Population size	125
Maximum # of pages	32
Page size	8 instructions
Maximum working page size	8 instructions
Crossover, Mutation, Swap probability	0.9, 0.5, 0.9
Tournament size	4
Number of registers	8
Instruction type 1, 2 or 3 probability	1/11, 8/11, 2/11
Function set	$\{+, -, \times, \div\}$
Terminal set	$\{0, ..., 255\} \cup$ {exemplar features}
Hierarchical Subset Selection Parameters	
Level 2 subset size	50
Max block selection iterations	1000
Max subset selection iterations (6 tournaments/iteration)	100
Level 1 block size	1000 (5000 KDD'99)

the four classification problems demonstrated no significant difference in the results. However, experimentation indicated that for imbalanced datasets it is beneficial to increase the distribution of the least represented class exemplars in the level 1 block compared to the original data distribution. In light of this observation, when using the BB-DSS algorithm for the following experiments the 37.5/62.5 partition ratio was chosen for comparison.

6 Results

6.1 Adult Dataset

Table 3 summarizes training and test performance over the various metrics. Overall it is apparent that both the RSS-DSS and CasGP algorithms have given more emphasis to FP rate minimization; whereas BB-DSS typically emphasizes maximization of detection rate. Under the distribution of major/minor class exemplars associated with the Adult dataset this BB-DSS policy results in an inferior overall error rate. Moreover, the performance of the CasGP algorithm is much more consistant, emphasizing the benefit of incrementally building solutions from multiple classifiers.

6.2 Census Dataset

Once again the CasGP algorithm has the lowest median error rates on both the training and test datasets (Table 4). The RSS-DSS algorithm, on the other hand, rarely achieves better than the degenerate error rate of approximately 5.66% on the test dataset (the degenerate error rate can be achieved by a solution labeling every exemplar as the major class). The BB-DSS algorithm has the tightest quartiles in terms of error rates and has medians that, although better than the RSS-DSS algorithm, do not approach that of the CasGP algorithm. T-

Table 3: Adult Results

	RSS-DSS	CasGP	BB-DSS
	Training Error		
1st Quartile	17.698	16.091	19.273
Median	18.891	16.284	21.378
3rd Quartile	20.718	16.524	23.016
	Test Error		
1st Quartile	17.839	16.212	19.170
Median	19.031	16.378	21.295
3rd Quartile	20.624	16.570	22.430
	Test DR		
1st Quartile	91.637	92.232	92.685
Median	92.694	92.742	95.845
3rd Quartile	97.364	93.418	96.347
	Test FPR		
1st Quartile	46.176	42.736	55.297
Median	51.541	44.689	71.541
3rd Quartile	73.378	45.757	78.919

tests for training and test error rates between the CasGP and BB-DSS algorithm return p-values of 8.57×10^{-5} and 4.97×10^{-4} respectively, therefore the results are statistically independent at the 95% confidence interval under both training and test conditions.

In summary, for the Census dataset the CasGP and the BB-DSS algorithm result in better test error rates than the RSS-DSS algorithm due to an improved detection rate distribution. However, the CasGP algorithm also appears to benifit from having a little better FP distribution thereby resulting in a better overall error rate on the Census dataset than the BB-DSS algorithm.

6.3 Shuttle Dataset

Table 5 summarizes performance on the Shuttle dataset. Both the CasGP and BB-DSS algorithms are significantly better than the RSS-DSS algorithm in terms of median error rate and consistency. For this dataset the CasGP algorithm had the best median error rate on the training dataset although this

Table 4: Census Results

	RSS-DSS	CasGP	BB-DSS
	Training Error		
1st Quartile	5.625	5.173	5.468
Median	5.788	5.252	5.512
3rd Quartile	5.978	5.356	5.591
	Test Error		
1st Quartile	5.707	5.099	5.292
Median	5.551	5.169	5.391
3rd Quartile	5.904	5.270	5.440
	Test DR		
1st Quartile	98.308	98.999	99.096
Median	98.960	99.312	99.312
3rd Quartile	99.547	99.512	99.522
	Test FPR		
1st Quartile	78.466	74.953	80.339
Median	81.327	79.240	83.340
3rd Quartile	88.744	82.864	86.974

did not translate into the best median error rate on the test dataset, which was returned by the BB-DSS algorithm. The CasGP algorithm was the most consistent algorithm in terms of training and test error rates.

All three algorithms show a very high median detection rate with the BB-DSS algorithm having the highest and the CasGP algorithm being the most consistent (Table 5). In terms of false positive rates the RSS-DSS algorithm fares the worst again, with the CasGP and BB-DSS algorithms being competitive in terms of median FPR and the BB-DSS algorithm being the most consistent.

On the Shuttle dataset the BB-DSS algorithm outperforms the RSS-DSS algorithm. Furthermore, the CasGP algorithm does not seem to improve on the results achieved by the BB-DSS algorithm. This results in the BB-DSS algorithm providing the best classifier for the Shuttle dataset.

Table 5: Shuttle Results

	RSS-DSS	CasGP	BB-DSS
	Training Error		
1st Quartile	0.957	0.312	1.081
Median	2.184	0.914	1.267
3rd Quartile	4.723	1.154	2.821
	Test Error		
1st Quartile	0.948	0.284	0.186
Median	2.214	0.917	0.493
3rd Quartile	5.019	1.084	2.183
	Test DR		
1st Quartile	97.761	99.124	97.687
Median	99.207	99.525	99.834
3rd Quartile	99.948	99.965	99.991
	Test FPR		
1st Quartile	0.728	0.695	0.670
Median	2.697	1.158	0.893
3rd Quartile	15.139	2.813	1.456

6.4 KDD'99 Dataset

Table 6 summarizes performance on the KDD'99 dataset. As on the Shuttle dataset, despite the CasGP algorithm having the lowest median error rate on the training dataset the BB-DSS algorithm has the lowest median test error rate. T-tests between BB-DSS and RSS-DSS for test error rates gives a p-value of 5.71×10^{-4} and between BB-DSS and CasGP a p-value of 6.50×10^{-13}, which indicates that there is a significant statistical difference between the algorithms' results. Also the RSS-DSS algorithm is competitive with the CasGP algorithm in terms of median test error rate. The CasGP and BB-DSS algorithms were more consistent than the RSS-DSS algorithm for both training and test.

Despite the CasGP algorithm having the lowest median error rate on the training dataset the BB-DSS algorithm provides the best classifier of the three algorithms on the KDD'99 dataset having the lowest median test dataset error rate and the lowest median false positive rate, which again is statistically significant with t-tests returning a p-value of 2.82×10^{-2} when compared to

Table 6: KDD'99 Results

	RSS-DSS	CasGP	BB-DSS
	Training Error		
1st Quartile	1.238	0.581	1.034
Median	1.553	0.641	1.201
3rd Quartile	2.285	0.758	1.238
	Test Error		
1st Quartile	8.631	8.675	7.641
Median	8.847	8.851	7.868
3rd Quartile	9.372	8.968	7.974
	Test DR		
1st Quartile	89.500	89.535	90.298
Median	90.077	89.730	90.491
3rd Quartile	90.301	89.915	90.763
	Test FPR		
1st Quartile	2.314	2.113	0.271
Median	3.662	2.757	0.807
3rd Quartile	5.937	3.402	1.124

RSS-DSS and 1.47×10^{-3} when compared to CasGP. Furthermore BB-DSS has the highest median detection rate which is statistically significant compared to CasGP with a t-test returning a p-value of 3.57×10^{-8}. Given the disparity between training and test performance it is possible that CasGP has overlearned the training data; whereas the partition based stopping criteria associated with the BB-DSS algorithm might decrease the likelihood of this.

6.5 Run Times and Degenerate Solution Frequencies

Table 7 shows the quartile run times for the BB-DSS algorithm on the Adult, Census and Shuttle dataset and the run times of the RSS-DSS, CasGP and BB-DSS algorithms on the KDD'99 dataset. On the KDD'99 datasets the CasGP algorithm clearly costs the most to train. However, a contributing factor to this is not so much GP as the routine used to append the training data with an additional 'feature' each time a new classifier is added to the cascade (the current routine uses a perl script). The BB-DSS algorithm was very effective

Table 7: 1st, 2nd (median) and 3rd quartile run times in minutes

Dataset	Adult	Census	Shuttle
Algorithm	BB-DSS		
1st Quartile	2.18	5.12	1.61
Median	4.52	10.63	2.53
3rd Quartile	8.84	10.70	5.07
Dataset	KDD'99		
Algorithm	RSS-DSS	CasGP	BB-DSS
1st Quartile	75.95	1071.50	20.09
Median	87.97	1121.11	28.36
3rd Quartile	102.88	1161.67	32.65

Table 8: Degenerates

	Adult	Census	Shuttle	KDD'99
RSS-DSS	6.6%	80%	0%	10%
CasGP	0%	0%	0%	0%
BB-DSS	3.3%	73.3%	0%	0%

on the Shuttle dataset, where the combination of an active learning algorithm and an early stopping criteria resulted in a median training time nearly twice as fast as on the Adult dataset despite there being 13,000 more exemplars in the Shuttle dataset.

Table 8 lists the total number of degenerate solutions found by each GP algorithm on each of the four datasets. The RSS-DSS and BB-DSS algorithms appear to have particular difficulty with the imbalanced Census dataset; however the BB-DSS algorithm is more consistant at avoiding degenerate solutions as a whole than RSS-DSS (lower degenerate counts over more datasets). The CasGP algorithm clearly fares the best in terms of avoiding degenerate solutions. We attribute this to the process of repeated error refinement over multiple layers as opposed to the one shot approach to learning taken by the RSS-DSS and BB-DSS algorithms. In addition, degenerate models that might exist in the cascade can be corrected by models added later in the architecture, thus not reflected in Table 8.

Table 9: Best Case Results on Adult dataset

Active Learning			
Model	RSS-DSS	CasGP	BB-DSS
Error	16.55	16.02	16.45
Canonical GP			
Node limit	256	1024	4096
Error	16.85	16.61	16.77
Time(hrs)	15.74	41.0	371.45

6.6 Canonical Tree GP without Active Learning

Results were also obtained on the adult dataset for a tree GP implementation (as opposed to linear GP) that does not use hierarchical subset selection. Specifically, the results were obtained using lilGP, which is an efficient C-based implementation of the original tree-structured GP [28]. Populations of 4000 individuals were used and the crossover, reproduction, and mutation probabilities were set to 90%, 10% and 0% respectively (i.e., Koza's canonical GP). The function set was limited to the case of arithmetic operators, as in the above results. Three cases were considered: the first used a node limit of 256, the second used a node limit of 1024 and the third used a node limit of 4096. This latter node limit corresponds to the case of a CasGP solution with 16 layers (each layer having an instruction limit of 256). Simulations were performed on an iMac desktop computer with a 1.25GHz G4 processor and 1 GB of RAM. For each case, ten runs of lilGP were performed and the results for the run with the lowest training error rate are shown in Table 9. The error rates are given in percent and training time is in hours. The extensive computational overhead limited the number of runs to ten per lilGP configuration.

The comparison provided by Table 9 indicates that GP algorithms with hierarchical subset selection provide competitive accuracies with canonical GP. In terms of run times the median BB-DSS results took less than 5 minutes (see Table 7) when compared to the 15 hours and above required for the standard GP

algorithm. Thus, the family of GP active learning algorithms are successfully able to decrease run time while maintaining competitiveness with canonical tree-based GP trained over the entire dataset.

7 Conclusion

This work benchmarks a family of Dynamic Subset Selection algorithms for scaling binary classifiers built under the GP paradigm to large unbalanced datasets without recourse to hardware specific speedups. The principle observation supporting this is that not all exemplars are created equal. Thus, as the current solution improves, then the number of exemplars of relevance for distinguishing between the performance of (GP) learners also decreases. Two heuristics are used to identify appropriate exemplars, age and difficulty. Age ensures that exemplars previously correctly classified are not forgotten, whereas difficulty is used to provide feedback on how frequently an exemplar is misclassified. Support for very large datasets is incorporated by introducing the concept of blocks of training data, where a block is small enough to reside in cache alone.

Three formulations of this hierarchical model are considered, the RSS-DSS algorithm, CasGP, and the Balanced Block DSS algorithm. The baseline RSS-DSS algorithm assumes an *a priori* relation between training exemplars and blocks, only utilizing the Dynamic Subset Selection algorithm to select suitable training subsets relative to the fixed block content. The CasGP algorithm uses the RSS-DSS algorithm to provide the basis for building cascades of GP classifiers, up to a predefined number of layers. The serial nature of constructing the CasGP model does, however, imply that the overall training time exceeds that for both RSS-DSS and the Balanced Block algorithms, but remains significantly faster than canonical GP.

The proposed Balanced Block algorithm results in a model based on a sin-

gle classifier, but introduces a more sophisticated methodology for building the blocks than the original RSS-DSS algorithm. This enables blocks and the ensuing Dynamic Subset Selection process to increase the representation of the minor class, whilst also carrying the concept of exemplar difficulty and age to the original dataset (both RSS-DSS and CasGP assume block selection of uniform probability). As such the Balanced Block algorithm is able to approach the classification performance of the CasGP algorithm, whilst retaining the computational speedup of the original RSS-DSS algorithm (i.e., a single classifier model). Key to achieving this was the partition based stop criterion of the BB-DSS algorithm, where this was able to successfully avoid over-learning on two of the datasets considered. A natural extension of the Balanced Block and cascade architectures would therefore be to replace the RSS-DSS algorithm in CasGP with the BB-DSS algorithm.

References

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

[2] C. Gathercole, and P. Ross, "Dynamic Training Subset Selection for Supervised Learning in Genetic Programming," in *Proc. 3rd Conf. Parallel Problem Solving from Nature*, Lecture Notes in Computer Science, vol. 866, pp. 312-321, 1994.

[3] D. Song, M. I. Heywood, and A. N. Zincir-Heywood, "A Linear Genetic Programming Approach to Intrusion Detection," in *Proc. Genetic and Evolutionary Computation Conference*, Lecture Notes in Computer Science, vol. 2724, pp. 2324-2336, 2003.

[4] D. Song, M. I. Heywood, and A. N. Zincir-Heywood, "Training Genetic Programming on Half a Million Patterns: An Example from Anomaly Detection," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 225-239, 2005.

[5] R. Curry, and M. I. Heywood, "Towards Efficient Training on Large Datasets for Genetic Programming," in *17th Can. Soc. Comp. Stud. Intel. Conf. Adv. Artificial Intelligence*, Lecture Notes in Artificial Intelligence, vol. 3060, pp. 161-174, 2004.

[6] P. Lichodzijewski, M. I. Heywood, and A. N. Zincir-Heywood, "Cascaded GP Models for Data Mining," in *IEEE Congress on Evolutionary Computation* vol. 2, pp. 2258-2264, 2004.

[7] P. Lichodzijewski, M. I. Heywood, and A. N. Zincir-Heywood, "CasGP: Building Cascaded Hierarchical Models using Niching," in *IEEE Congress on Evolutionary Computation*, vol. 2, pp. 1180-1187, 2005.

[8] G. Paris, D. Robilliard, and C. Fonlupt, "Applying Boosting Techniques to Genetic Programming," in *Int. Conf. Artificial Evolution*, Lecture Notes in Computer Science, vol. 2310, pp. 267-278, 2001.

[9] H. Iba, "Bagging, Boosting and Bloating in Genetic Programming," in *Proc. 1st Genetic and Evolutionary Computation Conference*, pp. 1053-1060, 1999.

[10] G. Folino, C. Pizzuti, and G. Spezzano, "GP Ensembles for Large-Scale Data Classification," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 5, pp. 604-616, 2006.

[11] C. Gathercole, and P. Ross, "Tackling the Boolean Even N Parity Problem with Genetic Programming and Limited-Error Fitness," in *Proc. 2nd Ann. Conf. Genetic Programming*, pp. 119-127, 1997.

[12] C. Lasarczyk, P. Dittrich, and W. Banzhaf, "Dynamic Subset Selection Based on a Fitness Case Topology," *Evolutionary Computation*, vol. 12, no. 2, pp. 223-242, 2004.

[13] P. Nordin, "A Compiling Genetic Programming System that Directly Manipulates the Machine Code," in *Advances in Genetic Programming*, Kinnear K.E. (ed.), Chpt. 14, Cambridge, MA: MIT Press, pp. 311-334, 1994.

[14] F.H. Bennett III, J. R. Koza, J. Shipman, and O. Stiffelman, "Building a Parallel Computer System for $18,000 that Performs a Half Petra-Flop per Day", in *Proc. Genetic and Evolutionary Computation Conference*, pp. 1484-1490, 1999.

[15] H. Julle, and J. B. Pollack, "Massively Parallel Genetic Programming," in *Advances in Genetic Programming 2* Chapter 17, P.J. Angeline, and K.E. Kinnear (eds.), Chpt. 17, Cambridge, MA: MIT Press, 339-358, 1996.

[16] J. R. Koza, F.H. Bennett III, J. L. Hutchings, S. L. Bade, M. A. Keane M.A., and D. Andre, "Evolving Computer Programs using Rapidly Reconfigurable Field Programmable Gate Arrays and Genetic Programming," in *Proc. ACM 6th International Symposium on Field Programmable Gate Arrays*, ACM Press, pp. 209-219, 1998.

[17] J. L. Hennessy, and D. A. Patterson, *Computer Architecture: A Quantitative Approach.* 3rd Ed., San Francisco, CA: Morgan Kaufmann, 2002.

[18] S. E. Fahlman S.E., and C. Lebiere, "The Cascade-Correlation Learning Architecture," in *Proc. Advances in Neural Information Processing Systems 2*, pp. 524-532, 1990.

[19] E. Litmann, and H. Ritter, "Cascade Network Architectures," in *Proc. IEEE-INNS Int. J. Conf. Neural Networks*, pp. 398-404, 1992.

[20] L. Huelsbergen, "Finding General Solutions to the Parity Problem by Evolving Machine-Language Representations," in *Proc. 3rd Conf. Genetic Programming*, San Francisco, CA: Morgan Kaufmann, pp. 158-166, 1998.

[21] M. Brameier, and W. Banzhaf, *Linear Genetic Programming: Genetic and Evolutionary Computation Series*, Berlin: Springer-Verlag, 2007.

[22] M. I. Heywood, and A. N. Zincir-Heywood, "Dynamic Page-Based Linear Genetic Programming," *IEEE Transactions on Systems, Man and Cybernetics - PartB: Cybernetics*, vol. 32, no. 3, pp. 380-388, 2002.

[23] G. Syswerda, "A Study of Reproduction in Generational and Steady-State Generational Algorithms," in *Foundations of Genetic Algorithms 1*, G.J.E. Rawlins (ed), San Francisco, CA: Morgan Kaufmann, pp 94-101, 1991.

[24] C. Elkan, "Results of the KDD'99 Classifier Learning Contest," *SIGKDD Explorations*, ACM SIGKDD, vol. 1, no. 2, pp. 63-54, 2000.

[25] D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz, "UCI Repository of machine learning databases," [http://www.ics.uci.edu/˜mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science, 1998.

[26] S. Hettich, and S. D Bay, "The UCI KDD Archive," [http://kdd.ics.uci.edu]. Irvine, CA: University of California, Department of Information and Computer Science, 1999.

[27] M. Wineberg, and S. Christensen, "Using Appropriate Statistics." in *Tutorial Program of the Genetic and Evolutionary Computation Conference*, J. Foster (ed), pp. 339-358, 2003.

[28] B. Punch, and E. Goodman, lilgp Genetic Programming System, v1.1 http://garage.eps.msu.edu/software/lil-gp/lilgp-index.html.