CasGP: Building Cascaded Hierarchical Models Using Niching

Peter Lichodzijewski, Malcolm I. Heywood, A. Nur Zincir-Heywood
Faculty of Computer Science,
Dalhousie University,
6050 University Avenue, Halifax, NS. B3H 1W5, Canada
{piotr, mheywood, zincir}@cs.dal.ca

***Abstract*— A Cascaded model is introduced for mining large datasets using Genetic Programming without recourse to specialist hardware. Such an algorithm satisfies the seeming conflicting requirements of scalability and accuracy on large datasets by incrementally building GP classifiers through the use of a hierarchical Dynamic Subset Selection algorithm. Models are built incrementally with each layer of the cascade receiving as input the original feature vector, plus the output from the previous layer(s). In order to encourage each layer to explicitly solve new aspects of the problem a combination of Sum Square Error and Niching is utilized. Thus, previous layers of the model are considered a niche, and the cost function is a shared error metric.**

I. Introduction

The principle interest of this work is to provide an efficient paradigm for mining large datasets using Genetic Programming (GP) i.e. a supervised learning context, without recourse to specialized hardware platforms. Previous works have suggested that an active learning algorithm such as Dynamic Subset Selection (DSS) may be used to address the computational overhead associated with large datasets [1]. Such a model decouples the computationally expensive inner loop of GP from the raw exemplar count by recognizing that not all exemplars are created equal. Thus, the dataset is filtered in accordance with an active learning algorithm. In this work we use the DSS methodology to provide the basis for efficient incremental learning on binary (as opposed to multi-class) data mining problems. The basic architecture takes the form of the hierarchical model popularized by the cascade correlation family of neural networks [2]. The ensuing models are modular, building on the results from previous layers such that each layer explicitly minimizes a new component of the error. Thus, the modular approach to problem solving provides for the decomposion of the problem into a hierarchy of different objectives. In order to avoid degenerate solutions we introduce a combination of sum square distance metric (as opposed to the count based metric typically employed in GP classification models) and a niche based fitness function. Such an approach is rather different from previous approaches to problem decomposition in GP, where the norm has been to build models in parallel and recombine using some sort of voting mechanism [3], [4], [5].

In the following we will first provide a short review of alternative schemes for encouraging problem decomposition, Section II, as well as introducing the basis for the DSS family of active learning algorithms utilized by this work. Section III provides the details of the CasGP algorithm, with a particular emphasis on the schemes used to minimize degenerate solutions at each layer of the hierarchy. Results are reported in Section IV for three benchmark problems. In particular a standard implementation of GP is used to establish the performance base line on several widely available data mining problems. This demonstrates that variable length GP is not able to compete with either the subset selection algorithm alone or CasGP; in effect code bloat results in significant computational overheads whilst model accuracy is also lacking. Moreover, comparison against other GP results based on ensemble methods indicates that CasGP is very competitive whilst avoiding the need for specialist hardware resources or reducing model transparency.

II. Related Work

As indicated above, previous works in which GP has explicitly supported problem decomposition has concentrated on some form Automatically Defined Functions [7], or more recently, have been based on ensemble methods such as Bagging/ Boosting. Indeed several instances of Bagging/ Boosting routines have appeared in GP. Specifically, a partitioned population model was utilized to construct ensembles of classifiers using both Bagging and Boosting by sampling [3]. This was then refined to produce Boosting by weighting [4]. Both schemes were demonstrated under small benchmark applications. In the case of larger datasets, a partitioned Bagging algorithm has recently been incorporated into a parallel cellular GP model, thus providing very fast training times [6]. Such schemes are based on the variance reduction methodology for model aggregation, where Bayesian Learning Theory predicts that multiple models will out perform a single model. However, this is not the only methodology by which multiple models may be combined to produce an aggregated model. Of particular interest to this work is the case of the cascade correlation architecture [2]. In this case models are not added in parallel, but hierarchically, with each layer of the hierarchy receiving input from all previous models and the original dataset i.e., each layer of the model adds a new feature to the input space. In addition, the original cascade-correlation scheme trained each new layer against the error residual, with adaptation only taking place in the layer currently under development. As a consequence, layer specific goals are explicitly identified.

Naturally, a hierarchical scheme for incremental model building requires an efficient methodology for constructing candidate models at each layer. In particular, training at each layer is still conducted over the entire dataset. However, the use of an active learning algorithm implies that the training data is filtered in accordance with the current classifier performance. To this end, use is made of the Dynamic Subset Selection (DSS) active learning algorithm [1], modified to incorporate the concept of computer memory hierarchies. The basic DSS algorithm collects two exemplar statistics, age and difficulty, and stochastically samples exemplars on this basis. Such a scheme naturally samples exemplars which are less frequently sampled or defeat classifiers more often. Computationally, the only drawback of such a scheme is that a random memory access model is still assumed. That is to say, on datasets larger than cache memory a significant penalty is encountered during cache misses. Previous work has addressed this case by first partitioning the original dataset into blocks, where a block of exemplars is sufficiently small to reside in cache memory alone [8]. Blocks are selected with uniform probability (random subset selection, or RSS) and the DSS algorithm applied to select exemplars within a block. More sophisticated block compositions and sampling algorithms have been considered for the hierarchy [9] and DSS itself [10], but in this work will retain the original RSS-DSS scheme of Song.

The CasGP model for hierarchical problem decomposition is independent of the specific form of GP employed. In this work a fixed length linearly structured GP representation is employed. Such a scheme shares many similarities with Genetic Algorithms in which alleles are allowed to take the form of a set of integers. For linear GP, the integers are decoded into instructions, typically taking the form of a register level transfer language. Programs are therefore ’run’ by starting at the first integer of an individual, decoding, executing and incrementing the ’Program Counter’ to point at the next instruction (integer). A register transfer language representation implies that instructions specify actions in terms of a predefined instruction set, target register, source registers, constants and inputs (features of the exemplar vector). Needless to say, varying constraints on the number of source registers specified results in different (register) addressing schemes, with instances of zero (stack) [11], two [12] and three [13] register addressing schemes having been demonstrated in a linear GP context. The fixed length representation used here implies that crossover always exchange an equal number of instructions. The initial population is therefore initialized with uniform probability over the entire range of program lengths, as opposed to the variable length scheme in which all individuals are initialized over a small subset of initial lengths. This work utilizes a steady state tournament in which children from the better performing half of the tournament replace the worse performing individuals from the same tournament. In addition the crossover scheme used here enforces an additional constraint, in which the location of crossover points also be fixed in an attempt to encourage code alignment between crossover points [12]. Mutation operators might be defined over specific instruction fields (e.g. opcode, source or target register) or applied across an entire instruction (the latter being used here). Moreover, it is also normal to incorporate a ’swap’ operator in which two instructions from the same individual are selected with uniform probability and interchanged.

III. CasGP Algorithms

In the following the three basic components of the CasGP architecture are introduced. The first component provides the basic algorithm for incrementally building hierarchically cascaded GP models and follows the basic error minimization methodology first identified for neural networks. Within the context of GP, this scheme frequently results in degenerate solutions [14]. That is to say, by adding the output from a previous layer of the cascade to the feature vector, the individuals at the GP population for the next layer quickly learn to copy this feature as their output. Naturally, this beats all other members of the population but does not further the performance of the overall cascade. Such a scheme is hereafter referred to as Naive CasGP. Two additional components are introduced to reduce the likelihood of degenerate solutions. The first step is to drop the use of a count based distance metric in favour of a sum square error, Subsection B. Finally, we introduce fitness sharing, where the basic objective is to explicitly penalize the duplication of error behaviour at the layer currently being constructed relative to previous layers. Subsection C details the construction of two fitness sharing functions.

A. Naive CasGP

The Naive CasGP algorithm is summarized in Algorithm 1. Each iteration of the outermost loop yields one layer of the cascade (i.e., one unit is added). The first layer of the cascade is trained with the original data and is equivalent to the generation of a single classifier. Each iteration of the algorithm then proceeds as follows. First, N different populations are initialized to mitigate the dependence of the algorithm on any one initialization (N of 1 and 30 are considered in Section IV). These populations are then trained on the data associated with the current layer. Once training is complete, the best run is identified as the one generating the fittest individual based on the training data. The data used to train the next layer is generated by augmenting the current layer’s data with the output of this best individual. Layers are added until a some termination criterion, such as a maximum depth (16 in the experiments of Section IV) or an acceptable error level, is reached.

B. Distance Metric and Wrapper

GP produces an output, typically real valued, and only limited by the numerical representation of the machine on which

Algorithm 1 Cascade model for incrementally building GP models.

```
do
{
initialize $N$ populations;
for ($i$ < $N$) train (data, pop($i$));
best = fittest(pop(0),..., pop($N$ - $i$));
cat(data, output(best));
}
until(termination);
```

it is evolved. In order to apply GP to (binary) classification problems the norm established by Koza has been to apply a binary mapping (or wrapper) to the original GP output [7]. This is synonymous with a switching function, centered at zero, Table I. Thus, any GP output smaller than zero (zero or greater) is associated with the majority (minority) class. An unfortunate consequence of such a decision is a reduction in the feedback provided to GP regarding the robustness of decision boundaries formed. That is to say, a solution in which exemplars from both classes are close to the switching point of the wrapper is treated the same as a solution in which a wrapper is placed between classes in such a way to establish a significant tolerance between class memberships. In effect switching type wrappers mask valuable distance information, without which it becomes increasingly difficult to provide robust classification models. Instead a mapping is required between the real valued GP output and the binary classification domain, such that distance information is retained. We address this problem by again borrowing concepts from neural network models. In particular a widely used mapping satisfying the above objective is the sigmoid or tansig function. Such a function monotonically maps a continuous input space to a unit interval or symmetric interval. The original (neural network) motivation for doing so was to provide a decision boundary that was smooth, thus differentiable. In the case of GP the smoothness constraint is redundant, however, the function does provide the basis for a more informative wrapper. Thus a 'good' placement of the wrapper would tend to force exemplars from dissimilar classes to opposite extremes of the sigmoid, where the means for making such a decision is provided by the monotonically increasing characteristic of the signmoid, Table I. Error or distance calculations now characterize how far GP is able to push exemplars into their respective classes as well as whether they are members of the relevant class (the switching wrapper only quantifies the latter). Moreover, the use of a real valued error (as opposed to a count of the number of correct classifications) implies that we may now utilize a cost function which explicitly penalizes errors in proportion to a predefined law. We previously demonstrated a preference for case of a sum square error (SSE), where this naturally assumes a Gaussian distribution of errors, Table I [14].

C. Fitness Sharing

The second approach for minimizing the potential for degenerate solutions across CasGP is based on the concept of fitness sharing originally demonstrated for (Genetic Algorithm) multi-modal optimization problems [15]. The basic problem faced in the multi-modal context was that the stochastic nature of the GA caused the solutions to converge to a single peak even when several peaks of equal fitness were present. The sharing approach tries to maintain diversity by reducing an individual's fitness when it is in close proximity to the solution identified by other individuals. In this way, the population is encouraged to disperse among the multiple optima. Informally, sharing reduces the fitness of an individual by a factor called the *niche count* which measures how similar an individual is to nearby individuals by way of an appropriate distance or *sharing function*.

TABLE I

WRAPPER AND ASSOCIATED DISTANCE METRIC.

Wrapper	Distance Metric
$y = \begin{cases} 1 \text{ if } GPout > 0, \\ 0 \text{ otherwise} \end{cases}$	$EC = \sum_p 1 - hit(y,p)$
$EC = \sum_p 1 - hit(y,p)$	$SSE = \sum_p (d_p - y_p)^2$

The concepts from sharing can be applied to the cascade architecture by considering the overlap in the performance of different layers. Thus, during the construction of the cascade, a distinction is made between the individuals that are in the process of being evolved at the top most layer and the solutions at the previous layers which have already been fixed. Only the fitness of the individuals being evolved are evaluated, and unlike Naive CasGP, this evaluation considers the behaviour of the solutions at the lower layers. In particular, the individuals being evolved are penalized if they reproduce the behaviour of (i.e., are similar to) previous layers. However, there is an important difference between this approach and sharing as traditionally defined. With traditional sharing, similarity is measured over both desirable and undesirable behaviours. Instead, with the cascade, individuals are penalized only when they make the same mistakes as previous layers. That is, individuals should be allowed to reproduce correct behaviour while focusing on correcting incorrect behaviour. Moreover, we also emphasize that by conducting sharing between a member of the current population and previously evolved solutions, the approach does not suffer from the problem associated with tournament selection [16]. That is, the values used for fitness sharing do not become outdated since they are not pre-calculated using other members of the population, but are based on the previously-fixed layers. In the following two such functions are defined, where smaller values correspond to fitter individuals i.e., a minimization objective.

1) S1 Sharing Function: In the case of sharing function S1, we let the shared fitness of an individual be the sum of the individual's raw fitness and normalized niche count. Raw fitness takes the form of the SSE cost function. Niche count penalizes erroneous behaviour by considering performance at previous layers. If the previous layer got the instance right, the niche count is incremented by the sum-squared error of the individual on that instance. If the previous layer got that instance wrong, the niche count is incremented by the product of the errors of the previous layer and the individual on that instance. When the niche count is added to the raw fitness, it is normalized by the number of layers to assure convergence at higher layers.

Specifically, the shared fitness of an individual *i* is defined to be,

$$f_{sh,i} = f_i + \frac{m_i}{L+1}$$

Here, *L* is the number of layers, f_i is the raw fitness, and mi is the niche count. Niche count is defined by,

$$m_i = \sum_{l \in L} \sum_{p \in T} sh\left(i, l, p\right)$$

with *T* corresponding to the the fitness cases. For an individual *i*, previous layer *l*, and fitness case *p*, the sharing function is defined as follows:

$$sh\left(i, l, p\right) = \begin{cases} 0 & \text{if } i \text{ got case } p \text{ right,} \\ (d_p - y_{i,p})(d_p - y_{l,p}) & \text{if } i \text{ and } l \text{ got case } p \text{ wrong,} \\ (d_p - y_{i,p})^2 & \text{if } i \text{ got case } p \text{ wrong but } l \text{ did not.} \end{cases}$$

In this definition, d_p corresponds to the desired value for fitness case *p*, while $y_{i,p}$ and $y_{l,p}$ corresponds to the values output for case *p* by individual *i* and layer *l* respectively. Note that the output values that are used have been transformed using the appropriate wrapper.

Thus, the sharing function does not penalize an individual if it classifies a fitness case correctly. If the individual classifies the fitness case incorrectly and the previous layer classifies the case incorrectly as well, then the niche count is incremented in proportion to how far off the two outputs were. Note that in this case $d_p - y_{i,p}$ and $d_p - y_{l,p}$ will always have the sign and their absolute values will be greater than 1, so their product will be greater than 1. In this way, the evolutionary process will not only penalize an individual based on the deviation of its output value from the desired value, but will penalize that individual more if previous layers had difficulty with that instance as well. Therefore, focus should shift to the more difficult cases.

If the individual classified the fitness case incorrectly and the previous layer classified that instance correctly, then the niche count is incremented by the squared deviation of the individual's output value from the desired value. In comparison, penalizing the individual as in the previous case would actually reward that individual for getting wrong what the previous layer got right since in this case $|d_p - y_{l,p}|$ is no larger than 1 while $|d_p - y_{i,p}|$ is necessarily larger than 1, so that $(d_p - y_{i,p})(d_p - y_{l,p})$ is less than $(d_p - y_{i,p})^2$. As is, the sum-squared error on that fitness case is essentially counted twice as penalty for misclassifying an instance that another layer got right.

In the final definition of the shared fitness the niche count is normalized by the number of layers. This was found to be necessary in order for convergence to take place. Otherwise, at higher layers, it was observed that many different but poor solutions were evolved. This was attributed to the fact that without the normalization factor, the weight of the niche count in contribution to the final shared fitness would steadily increase as more layers were added. Thus, focus shifted from generating good and unique solutions to generating only unique solutions.

2) S2 Sharing Function: At the first layer, when there are no previous layers to compare the current individual against, the fitness of an individual is evaluated using the SSE cost function. At higher layers, the shared fitness is calculated by iterating over all the previous units, and for each previous unit, incrementing the shared fitness by the error of the individual with respect to that unit. If the distance to the given unit is sufficiently large, the error is added to the shared fitness as is. Otherwise, the individual and the previous unit are deemed to be too similar, and the weight of the error is increased in proportion to the distance. The distance used is a Euclidean distance based on the values that the current individual and previous unit output for each fitness case. The error with respect to each previous unit, again based on the fitness cases, is affected only when the deviation from the desired output of the current individual is greater than the deviation of the previous unit. In this case, the error is increased by the square difference of the two deviations.

More formally, beyond the first layer, the shared fitness of an individual *i* is defined as,

$$f_{sh.i} = \sum_{l \in L} (1 + e_{i,l})(1 + sh(d_{i,l}))$$

Here, L again corresponds to the previous layers, $e_{i,l}$ is the error for individual *i* associated with layer *l*, $d_{i,l}$ is the distance between individual *i* and layer *l*, and the sharing function is defined as:

$$sh(d_{i,l}) = \begin{cases} 1 - \left(\frac{d_{i,l}}{\sigma_{sh}}\right)^{\alpha_{sh}} & \text{if } d_{i,l} < \sigma_{sh} \\ 0 & \text{otherwise} \end{cases}$$

As with traditional sharing, σ_{sh} is the neighbourhood radius and α_{sh} controls the shape of the function. The distance $d_{i,l}$ is the Euclidean distance evaluated over the values output by *i* and *l* for each of the fitness cases, that is,

$$d_{i,l} = \sqrt{\sum_{p \in T} (y_{i,p} - y_{l,p})^2}$$

Finally, the error of individual *i* associated with layer *l* is defined as,

$$e_{i,l} = \sum_{p \in T} se(i, l, p)$$

where

$$se(i, l, p) = \begin{cases} ((d_p - y_{i,p}) - (d_p - y_{l,p}))^2 \, if \; |d_p - y_{i,p}| > |d_p - y_{l,p}|, \\ 0 \quad \text{otherwise.} \end{cases}$$

Note that in the above definition $|d_p - y_{i,p}|$ and $|d_p - y_{l,p}|$ correspond to the deviations from the desired value of the values produced by *i* and *l* for pattern *p*.

The S2 sharing approach addresses two specific issues. First, individuals are encouraged not to perform worse than previous layers since the calculation of $e_{i,l}$ counts error only if it is greater than that of the previous layer. With this alone, individuals would minimize error by reproducing exactly the outputs of previous layers, and once again, degenerates would be produced. To prevent this from happening, the approach penalizes individuals if, based on their output values, they are similar to previous layers. This should lead to higher layers producing individuals that are no worse than those in the lower layers but yet remain different. The ultimate goal being to identify individuals that would classify instances that previous layers got wrong, units at the top of the cascade could then use the outputs of the lower layers to generate very accurate classifiers. This approach is more similar to the original formulation of fitness sharing.

IV. Results

In the following performance is considered across three benchmark classification datasets: Adult, Shuttle, and Census. Evaluation is performed by considering cascades built using one of two methodologies. In both cases 30 initializations are considered, however, in the first scheme 30 runs are performed using 1 initialization per cascade layer, resulting in 30 different cascades. This hereafter is referred to as CasGP-30. The second architecture performed 30 different initializations per layer, selecting the fittest (on training data) to represent that layer, resulting in a single cascade, or CasGP-1. The following subsections summarize characteristics of the four benchmarks and deal with the parameterization employed throughout for CasGP before beginning the performance comparison itself.

A. Datasets

Only binary classification problems are considered in this work. The three benchmark data mining datasets considered here are originally binary or converted to binary classification problems as detailed below.

1) Adult: The Adult dataset was obtained from the UCI Machine Learning Repository. Each exemplar in the dataset has a feature length of 14, representing a mixture of continuous and nominal features. The task was to decide whether a person earned over $50 000 per year. The version of the dataset used did not contain unknown values. This yielded a total of 30 162 training and 15 060 test instances. Of the 45 222 total instances, 24.78% corresponded to individuals that earned over $50 000 per year. In preprocessing the data, continuous features were left unchanged whereas is nominal features were mapped to integers.

2) Shuttle: The Shuttle dataset that was used was based on the Shuttle dataset found in the Statlog database, also found in the UCI Machine Learning Repository. It represents conditions and desired actions for controlling a space shuttle. The original version of the dataset contained 43 500 training and 14 500 test instances each composed of nine numerical features. Each instance belonged to one of seven classes, although class 1 accounted for most of the instances. The problem was converted to a binary classification problem by labeling class 1 as negative and all other classes as positive. This resulted in 21.40% of instances belonging to the positive class.

3) Census: Similar to the Adult dataset, the objective is to classify whether a person earned over $50 000 per year. In this case, however, there are 40 features (as opposed to 14 in the case of Adult) describing demographic and personal characteristics of each exemplar. The resulting dataset consists of 95 130 training and 47 391 test exemplars, with approximately 5.6% of the data representing the minor class (on both partitions). This latter characteristic makes the dataset an interesting test of performance under very unbalanced conditions. The dataset was also sourced from the UCI repository.

B. GP Parameterization

As indicated in the introduction, one of the motivations for designing CasGP was as an alternative to variable length GP. The CasGP algorithm, however, is actually independent of the form of GP employed. The fix length GP employed throughout these experiments is denoted page-based Linear-GP with dynamic crossover [12]. The parameters used are detailed in Table II. Three general points are note worthy. First, the maximum initial number of pages and the initial page size are used to define the size of the individuals at initialization. Given the settings in Table II, the maximum individual size is 32 × 8 = 256 instructions. Second, a maximum page size of 8 means that the page sizes dynamically vary between 1, 2, 4, and 8, all within the same training run as training error plateau's are encountered. Third, the weights associated with instruction types define the expected proportion of each instruction type in the initial population. Three instruction types are: Type 1, load register with a constant; Type 2, apply opcode to an input and register; or Type 3, apply opcode to two registers. Thus, given the weights shown, the most common instruction should involve a register and an external input.

TABLE II

GP and RSS-DSS Parameterization

Page Based Linear GP	
Parameter	Value
Population Size	125
Max Page Size	8 Instructions
Max Pages	32
Number of Registers	8
Crossover probability	0.9
Mutation probability	0.5
Swap probability	0.9
Probability of initializing with Type 1, 2 or 3 instruction	1/11, 8/11, 2/11
Tournament Size	4
Function Set	$\{+, -, \times, \div\}$
Terminal Set	$\{0, ..., 256\} \cup$ {input features}
RSS-DSS Parameters	
RSS Block Size	5 000
DSS Subset Size	50
RSS Iterations	$1000 \times \lfloor (blocks + 5) / 10 \rfloor$
Max DSS Iterations	100
Prob. of DSS pattern selection using age (difficulty)	0.3 (0.7)
DSS Refresh Frequency	6

To facilitate training with large datasets, the page-based L-GP with dynamic crossover was combined with the hierarchical RSS-DSS algorithm [8]. The parameters associated with the RSS-DSS algorithm are shown in Table II. Two things are of note. First, the number of RSS selections is defined in proportion to the number of blocks used to partition the training dataset; if the dataset has more blocks, more RSS selections are made. The idea is that, on average, the number of times the entire dataset is traversed will remain constant regardless of the number of training exemplars. Secondly, the actual number of DSS selections made per tournament depends on the difficulty of a block. This will decrease as classifier performance improves, thus concentrating the fitness evaluation on the most pertinent exemplars i.e., difficult or old, as per the DSS algorithm. Detailed descriptions of the Dynamic Subset Selection algorithms and associated analysis of parameterization are available elsewhere [1], [8].

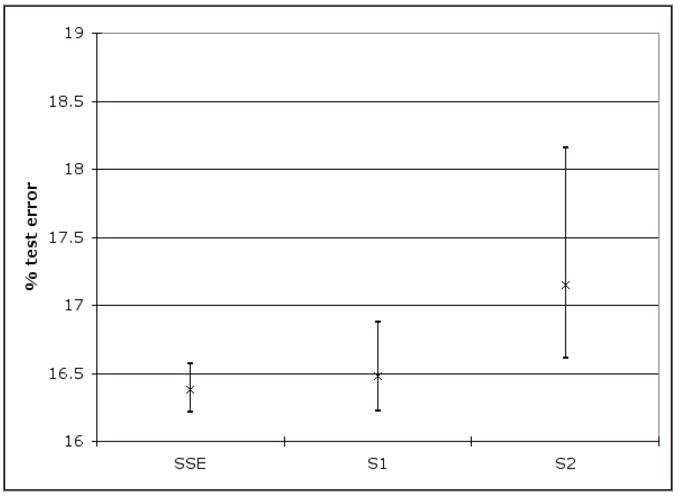

Fig. 1. CasGP-30 Quartile % Errors on Adult test data

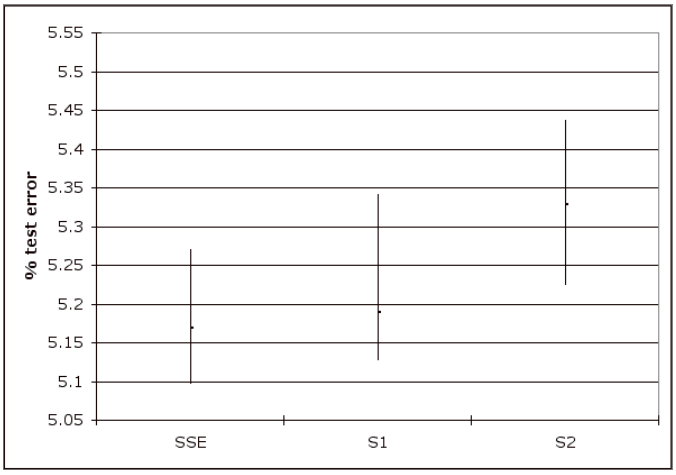

Fig. 2. CasGP-30 Quartile % Errors on Census test data

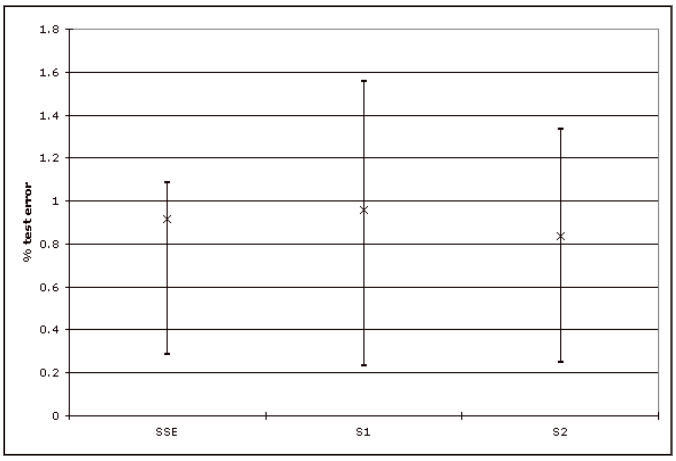

Fig. 3. CasGP-30 % Quartile Errors on Shuttle test data

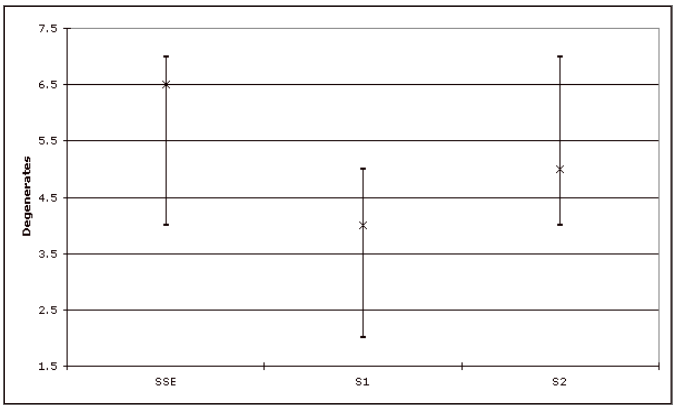

Fig. 4. CasGP-30 Quartile Degenerate Count on Adult dataset

C. Evaluation

As indicated above in the following we will consider two schemes for building cascaded hierarchies: CasGP-30; and, CasGP-1. In the following use will therefore be made of both best case and quartile performance, as well as results previously published for the various datasets in question.

1) CasGP-30 Hierarchies: The CasGP-30 methodology results in a new classifier for each run, but only performs one initialization per layer of the cascade. This means that we are able to provide some qualification of the degree of spread associated with the resulting cascades. Conversely, as the CasGP-1 architecture selects the best classifier from 30 initializations per layer, the scheme is less dependent on initial conditions than CasGP-30, but will only produce a single classifier from the same number of initializations.

In the case of CasGP-30, classification performance is reported for the layer with lowest training error (not necessarily the last layer). Figures 1 to 3 detail the test error on Adult, Census, and Shuttle datasets respectively (training curves showing similar characteristics). It is apparent that the sharing algorithms in this case do not result in a improvement above that provided using SSE alone. SSE actually results in a lower test error, as measured by a *t*-test at the 99% confidence interval for Adult and Census with respect to the S2 sharing function. Moreover, S2 tends to utilize more layers before the best training error was identified than either S1 or SSe, Table III. The principle benefit from using the shared fitness functions on CasGP-30 appears to have been in terms of the degenerate counts (number of layers at which a classifier just copied the result from the previous layer), Figures 4 to 6. Thus, S1 returns less degenerates on Adult and S2 less degenerates on Shuttle and Census, where this is again significant at the 99% confidence interval as measured by a *t*-test.

2) Previous Results and CasGP-1: Tables IV and V summarize best case results for the CasGP algorithms (using training performance to select best cases for CasGP-30). Needless to say, the CasGP-1 results are uniformally better than those returned under CasGP-30, where the CPU time

TABLE III

CASGP-30 MEDIAN TOTAL CASCADE LAYERS

Dataset	SSE	S1	S2
Adult	10	11	12
Census	9.5	11	12
Shuttle	5	6	9.5

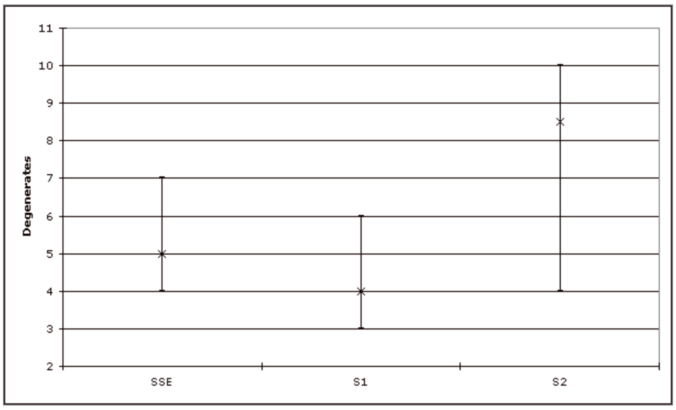

Fig. 5. CasGP-30 Quartile Degenerate Count on Census dataset

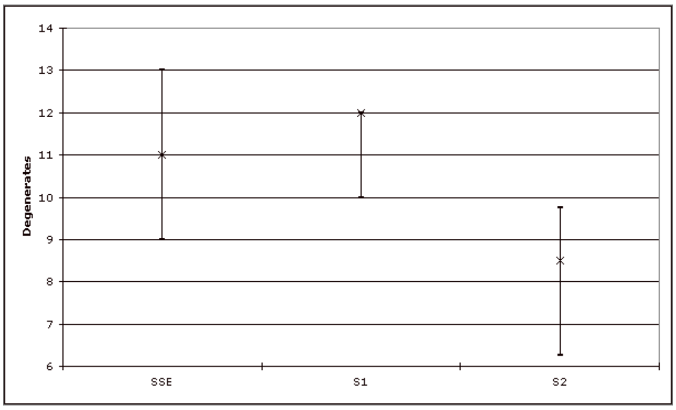

Fig. 6. CasGP-30 Quartile Degenerate Count on Shuttle dataset

requirement is the same for both schemes (both use a total of 30 initializations overall). This result is confirmed by a *t*-test returning p values of 0.002, 0.029 and 0.57 for Adult, Census and Shuttle respectively. In contrast to the general findings of the quartile CasGP-1 results above, it is the solutions identified using the sharing functions that result in the best results, irrespective of dataset.

In the case of the Census dataset, results were available from the UCI repository for decision tree algorithms and a Naive-Bayes Classifier, Table VI, with performance of both CasGP-1 and CasGP-30 remaining competitive with the decision trees, and bettering that of the Naive-Bayes Classifier. On the Adult and Census dataset previous results from a GP solution based on a distributed parallel Cellular Automata, Table VII, demonstrate that the median CasGP-30 results, Figures 1 and 2, match those reported for the Cellular Automata using 10 fold cross validation [6].

TABLE IV
% BEST CASE CASGP-30 ACCURACY

	Training			Test		
Dataset	SSE	S1	S2	SSE	S1	S2
Adult	15.91	15.64	16.09	16.56	15.88	16.02
Census	5.05	5.0	4.96	4.89	4.9	4.82
Shuttle	0.14	0.14	0.156	0.145	0.131	0.145

TABLE V
% BEST CASE CASGP-1 ACCURACY

	Training			Test		
Dataset	SSE	S1	S2	SSE	S1	S2
Adult	15.49	15.31	15.29	15.7	15.52	15.47
Census	4.98	4.89	4.87	4.9	4.85	4.8
Shuttle	0.06	0.05	0.14	0.1	0.14	0.13

TABLE VI
BEST CASE TEST RESULTS FOR DECISION TREE AND NAIVE-BAYES CLASSIFIERS ON CENSUS DATASET

Classifier	% Test Error
C 4.5	4.8
C 5.0	4.7
C 5.0 Rules	4.7
C 5.0 Boosting	4.6
Naive Bayes	23.2

As a final test, 10 runs were made using (variable length) tree structured GP without the RSS-DSS algorithm (fitness was evaluated across all training exemplars) using the Adult dataset. Parameterization follows the basic Koza utilization of generational selection, a population of 4 000 individuals and 90% crossover (10% mutation). Fitness is evaluated using a SSE cost function. Figure 7 plots test performance as quartiles for CasGP-30 and the vanilla implementation of tree structured GP (implemented using lilgp [17]). The benefit of SSE and S1 based CasGP-30 cascades is now readily apparent, with the S2 sharing function resulting in more sensitivity to initial conditions under this dataset. Moreover, all the cascade solutions required an order of magnitude lower node count than that returned for the variable length tree representation, Table VIII.

V. CONCLUSION

The Hierarchical Cascade architecture has been introduced for efficiently building modular solutions to large classification datasets. Specific attention is paid to the use of a suitably informative cost function and the number of initializations performed per layer. Evaluation of schemes based on one

TABLE VII
AVERAGE % TEST ERROR USING PARALLEL CELLULAR GP ON ADULT AND CENSUS DATASET

Classifier	Adult	Census
CGPC	17.18	5.19
BagCGPC	17.01	5.08
BoostCGPC	16.53	8.33

TABLE VIII
NODE COUNTS FOR VARIABLE LENGTH TREE STRUCTURED GP AND CASGP-30 ON ADULT DATASET

Quartile	Tree GP	SSE	S1	S2
3rd	3219	424.25	466.25	495.5
Median	2604	379.5	390.5	418
1st	1519.5	316.5	353	385

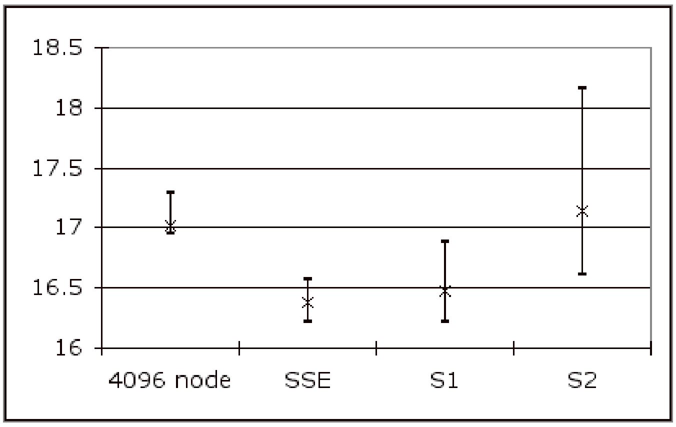

Fig. 7. % Test Error on Adult for variable length Tree structured GP and CasGP-30

initialization per layer over 30 runs (CasGP-30) versus 30 initializations per layer for a single run (CasGP-1) demonstrates that the best performance is still dominated by considering multiple initializations per layer. However, the performance of the CasGP-30 is still competitive with that returned by a parallel Cellular GP implementation with either bagging or boosting. This is achieved without the requirement for the necessary parallel hardware, albeit taking several hours to return the same level of performance. We note, however, that both CasGP-1 and CasGP-30 readily support parallel implementations.

Comparison with traditional variable length GP on the Adult dataset indicates that the resulting solutions are indeed far more succinct (lower node count) without impacting on the classification performance. Moreover, by using fixed sized individuals, but using the cascade to provide the basis for variable length solutions, we maintain that the cascade architecture is much better at incrementally decomposing the problem at each layer. This naturally does not preclude using variable length chromosomes, though whether this would result in radically different behaviour from that observed here is an open question.

The Sharing functions, S1 and S2, appeared to be more successful in identifying best case solutions than providing consistent classification performance. Thus the CasGP-1 architecture returned better results using the sharing functions than the SSE cost function alone (best of 30 initializations selected from at each layer). On CasGP-30, however, both S1 and S2 did provide a reduction in the number of degenerate solutions across the cascade, where this is significant at the 99% confidence interval. Future work continue to identify whether this is endemic to the formulation of the cost function or a question of parameterization.

References

[1] C. Gathercole, P. Ross, "Dynamic training subset selection for supervised learning in genetic programming." In Parallel Problem Solving from Nature III, Lecture Notes in Computer Science, LNCS 866, Springer-Verlag, pp 312-321, 1994.

[2] S. E. Fahlman and C. Lebiere., "The cascade-correlation learning architecture." In Proceedings of Advances in Neural Information Processing Systems-2, Morgan Kaufmann, pp 524-532, 1990.

[3] H. Iba, "Bagging, Boosting and Bloating in Genetic Programming." Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-99. Banzhaf W., et al. (Eds), Morgan Kaufmann. pp 1053-1060, 1999.

[4] G. Paris, D. Robilliard, C. Ronlupt, "Applying Boosting Techniques to Genetic Programming." 5th International Conference on Artificial Evolution. Collet P., et al. (Eds), Lecture Notes in Computer Science, LNCS 2310. pp 267-278, 2001.

[5] M. Brameier and W. Banzhaf. "Evolving teams of predictors with linear genetic programming." Genetic Programming and Evolvable Machines, 2(4), pp 381-407, 2001.

[6] G. Folino, C. Pizzuti, G. Spezzano, "Boosting Technique for Combining Cellular GP Classifiers," Proceedings of the 7th European Conference on Genetic Programming, EuroGP'04, Lecture Notes in Computer Science, LNCS 3003, Springer-Verlag. pp 47-56, 2004.

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

[8] D. Song, M.I. Heywood, A.N. Zincir-Heywood, "Training Genetic Programming on Half a Million Patterns: An Example from Anomaly Detection," To appear in: IEEE Transactions on Evolutionary Computation, 2005.

[9] R. Curry, M.I. Heywood, "Towards Efficient Training on Large Datasets for Genetic Programming," Advances in Artificial Intelligence, Proceedings of the 17th Conference of the Canadian Society for Computational Intelligence. A. Y. Tawfik, S. D. Goodwin (eds), Lecture Notes in Computer Science, LNCS 3060 Springer-Verlag. pp 161-174, May, 2004.

[10] C. Lasarczyk, P. Dittrich, and W. Banzhaf. "Dynamic subset selection based on a fitness case topology." Evolutionary Computation, 12(2), pp 223-242, 2004.

[11] T. Perkis, "Stack Based Genetic Programming." Proceedings of the IEEE Congress on Computational Intelligence. IEEE Press, 1994.

[12] M.I. Heywood, A.N. Zincir-Heywood, "Dynamic Page Based Crossover in Linear Genetic Programming," IEEE Transactions on Systems, Man and Cybernetics - Part B, 32(3), pp 360-388, June 2002.

[13] J.P. Nordin, W. Banzhaf, "Evolving Turning Complete Programs for a Register Machine with Self-Modifying code." In Proc. 6th International Conference on Genetic Programming. MorganKaufmann, pp 318-325, 1995.

[14] P. Lichodzijewski, M.I. Heywood, A.N. Zincir-Heywood, "Cascaded GP Models for Data Mining," IEEE Congress on Evolutionary Computation, CEC-04, Portland, Vol.2, pp 2258-2264, 2004.

[15] K. Deb and D.E. Goldberg, "An Investigation of niche and specis formation in genetic function optimization." Proceedings of the 3rd International Conference on Genetic Algorithms. pp 42-50, 1989.

[16] B. L. Miller and M. J. Shaw. Genetic algorithms with dynamic niche sharing for multimodal function optimization. Technical Report 95010, IlliGAL, University of Illinois at Urbana-Champaign, 1995.

[17] B. Punch and E. Goodman. lil-gp genetic programming system, v.1.1 [http://garage.cps.msu.edu/software/lil-gp/lilgp-index.html], Genetic Algorithms Research and Applications Group, 1998.