Pareto-Coevolutionary Genetic Programming for Problem Decomposition in Multi-Class Classification*

Peter Lichodzijewski and Malcolm I. Heywood†

July 10, 2007

Abstract

A bid-based approach for coevolving Genetic Programming classifiers is presented. The approach coevolves a population of learners that decompose the instance space by way of their aggregate bidding behaviour. To reduce computation overhead, a small, relevant, subset of training exemplars is (competitively) coevolved alongside the learners. The approach solves multi-class problems using a single population and is evaluated on three large datasets. It is found to be competitive, especially compared to classifier systems, while significantly reducing the computation overhead associated with training.

1 Introduction

Problem decomposition within the context of Genetic Programming (GP) [10] has focused mostly on module acquisition. That is to say, the principal interest has been to identify and propagate reusable code fragments throughout the population so that solutions need not repeatedly rediscover essential 'building blocks' [11]. In this work, in contrast, we are interested in problem decomposition by means of multiple individuals that learn to participate under different exemplar conditions.

Within the context of GP applied to the classification domain, recent examples of this second form of problem decomposition include teams [4], multi-objective optimization [13], distributed boosting [9], and Pareto-based competitive coevolution [12]. The teams metaphor [4] explicitly initializes a set of classifiers to coexist and evaluates their performance using a predefined voting scheme. The multi-objective approach [13] encourages classifiers to act as novelty detectors as opposed to the more typical discriminator approach to classification. Its main drawback is that it requires a clustering algorithm to

*Published in GECCO'07, ACM 978-1-59593-697-4/07/0007.

†Faculty of Computer Science, Dalhousie University, Halifax, Nova Scotia, Canada.

be applied within the inner loop of GP making it computationally expensive. Distributed boosting [9] requires a Beowulf computing platform for effective operation, evolves independent populations on partitions of the training data, and requires classifiers labels to be resolved post-training using a voting policy. Finally, the competitive Pareto-Coevolutionary GP Classifier (PGPC) [12] provides a solution in the form of an archive of non-dominated classifiers, however, it again requires a post-training voting scheme to provide a single class label under testing conditions.

The approach taken in this work is rather different. When individuals are first created they are bound to a static scalar selected over the range of possible class labels. As a result, an instance class may be represented by more than one individual. The goal of each GP individual is then to evolve an appropriate bidding behaviour. For each exemplar, the individuals determine their respective bids with the maximum bid 'winning' the right to present its class label. Classification error is estimated with respect to the suggested and actual class labels over all exemplars. Thus, in any one training epoch, the overall classification error reflects contributions from all individuals winning a bidding round. As such, the model provides a natural mechanism for coevolving a solution from multiple individuals without the need for a post-training voting scheme. The number of individuals participating in the final solution also follows naturally and does not need to be specified *a priori*.

This work also considers how to integrate the bid-based cooperative mechanism with a Pareto-based competitive coevolutionary paradigm [6, 8, 14]. In particular, the competitive coevolutionary paradigm models the interaction between learners (bidding individuals in this case) and test points (training exemplars) as a competitive game. Using the proposed approach, it is no longer necessary to iterate over all the training exemplars in order to evaluate learner fitness, but rather, over some subset of exemplars that maximize the distinctions between the learners. Such a model has been demonstrated to be significantly better at avoiding degenerate solutions on large unbalanced datasets while requiring a fraction of the computational overhead [12].

Evaluation of the proposed approach is done over datasets whose sizes range from few thousand to half a million exemplars and which involve up to five classes. Performance is evaluated in terms of a baseline model in which the competitive coevolutionary model is replaced with uniform random selection of training exemplars. On all the datasets, the model requires training times in the region of one to two and a half hours to successfully compose solutions using multiple learners per class.

The rest of this paper is organized as follows. The following section reviews related work including ideas on using coevolution for ideal evaluation. Section 3 describes in detail the proposed approach. The system is evaluated in Section 4 and conclusions are drawn in Section 5.

2 Related Work

2.1 Learning Classifier Systems

The proposed system shares similarities with Learning Classifiers Systems [3]. However, the various forms of classifier systems typically evolve condition-action-strength rules while the proposed approach replaces condition and strength with a single bid procedure. This results in deterministic individual behaviour because, in contrast to strength, bid values are not adapted during an individual's lifetime. Although the proposed approach does borrow from classifier systems in using accuracy as a component of individual fitness [16], the way in which this accuracy affects the individual's survival, and in general the proposed training algorithm, is very different. Finally, the proposed approach, at least for now, is restricted to supervised learning tasks, whereas classifier systems can be applied to reinforcement learning problems.

2.2 Ideal Evaluation and Pareto-Coevolution

The concepts of Pareto-dominance and distinctions [8, 14] are used in the Delphi system [6] to approximate an ideal evaluation function using a computationally feasible number of tests. This coevolutionary framework evolves a population of learners against a population of tests. The approach centers around an interaction function $G(l_i, t_k)$ which returns values from an ordered set and indicates the outcome of applying learner l_i to test t_k (revealing information about the underlying problem objectives).

If $\vec{v}_1$ and $\vec{v}_2$ are two objective vectors then the Pareto-dominance relation $dom(\vec{v}_1, \vec{v}_2)$ which indicates that $\vec{v}_1$ dominates $\vec{v}_2$ is defined as:

$$dom(\vec{v}_1, \vec{v}_2) \Leftrightarrow \forall q : \vec{v}_1[q] \geq \vec{v}_2[q] \wedge \exists q : \vec{v}_1[q] > \vec{v}_2[q]. \tag{1}$$

Delphi defines an objective or outcome vector for the ith learner l_i over all the tests as

$$\vec{o}_{l_i}[k] = G(l_i, t_k) \tag{2}$$

where t_k is the kth test. The Pareto-dominance relation applied to these outcome vectors, referred to as the coevolutionary evaluation function, is then used to determine which learners are discarded from the population.

If the number of learners is n, the approach also constructs an n^2-dimensional distinction vector $\vec{d}_{t_k}$ for each test t_k as

$$\vec{d}_{t_k}[n \cdot i + j] = \begin{cases} 1 & \text{if } G(l_i, t_k) > G(l_j, t_k) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where $1 \leq i, j, \leq n$ index the learners. Hence, a distinction is made if the outcome on t_k for learner l_i is *strictly* greater than the outcome for learner l_j. On the test side, these distinction vectors are used to determine which tests are discarded from the test population again using Pareto-dominance.

An ideal evaluation function, which determines whether learner l_i is superior to learner l_j, is defined using the Pareto-dominance relation on the underlying objectives. With the above setup and a test population of size $n^2 - n$, the coevolutionary evaluation function is capable of representing this ideal evaluation function *exactly* because $n^2 - n$ is an upper bound on the number of distinctions that can be made. Such a Complete Evaluation Set may not be necessary for successful coevolution and the size of the test population relative to the size of the learner population may be reduced further. This serves as motivation for this work where the goal is to reduce the GP training overhead.

3 Methodology

As in the Delphi system, a test population and a learner population are coevolved. The test population is a subset of the entire training dataset[1]. The learner population consists of a set of bidders whose behaviour is evolved using GP. Specifically, each learner defines a *bid* and an *action*. The bid is represented as a program while the discrete action is selected *a priori* from the set of possible instance labels.

To classify a test exemplar post-training each learner executes its bid program on the input feature vector and submits an associated bid. The highest bidder is selected as the winner and the action of this winner is then output as the label for the test exemplar. Together, the goal of the learners is to correctly classify the tests. The goal of the tests is to accurately distinguish between the learners.

3.1 Coevolutionary Training Algorithm

Unbalanced class distributions may adversely affect training by favouring learners based only on their action. For this reason, the learner population is partitioned so that selection and search operators are applied to learners of the same class. In addition, since distinctions between learners of different classes are not meaningful in this setup, a separate test subpopulation is maintained for each learner partition. This limits interactions to the learners within each partition and to learners in each partition and the tests in the corresponding test subpopulation. To simplify implementation the learner populations can therefore be trained in series[2] and this is the approach taken here, Figure 1.

The algorithm outputs a solution set of learners S which is initialized to be empty in Step 1. Training then proceeds on a class by class basis, Step 2. The learner population L_{pop} is initialized with random bid programs and action matching the current class a under consideration, Step 2a, and the test population T_{pop} is set to contain non-duplicated indices into the training dataset

[1] As opposed to the test data employed for assessing the generalization performance of the trained classifier.

[2] Compared to a parallel implementation, training overhead increases minimally as Step 2(c)ii in Figure 1 is executed $|L|$ times more often where L is the set of class labels.

1. $S = \emptyset$
2. for each $a \in L$
   (a) initialize(L_{pop}, a)
   (b) initialize(T_{pop})
   (c) for each epoch
      i. L_{pop} = generate(L_{pop})
      ii. T_{pop} = generate(T_{pop})
      iii. for each $t_k \in T_{pop}$
         A. bid(L_{pop}, t_k)
      iv. normalize(L_{pop})
      v. L_{pop} = select(L_{pop})
      vi. T_{pop} = select(T_{pop})
   (d) L_{pop} = reduce(L_{pop})
   (e) $S = S \cup L_{pop}$

Figure 1: Generic coevolutionary training algorithm. S is the final solution output by the system, L_{pop} and T_{pop} are the learner and test populations, and L is the set of exemplar labels.

selected with uniform probability, Step 2b. The learner and test populations are then evolved in a series of epochs, Step 2c. Here, in contrast to the *non-candidate* learners which already exist at the beginning of each epoch, *candidate* learners refers to the new members generated in Step 2(c)i (analogous terminology is used for tests). Once training of the current learner population is complete, some individuals in the populations may be removed by means of a reduction procedure, Step 2d. The learners in the reduced population are added to S in the final step, Step 2e.

At the start of each epoch L_{pop} and T_{pop} are augmented with candidate learners and tests, Steps 2(c)i and 2(c)ii respectively, and then the learners are applied to each test in a series of dummy bidding rounds, Step 2(c)iiiA. During a dummy bidding round for a test t_k each learner has its outcome on t_k calculated and recorded in an associated outcome vector. Once the bidding rounds are completed, the outcome vectors are normalized in Step 2(c)iv. Using these normalized outcome vectors, the algorithm determines if a candidate learner should remain in the population, Step 2(c)v. A candidate learner can remain in the population only by displacing an existing non-candidate learner to which it is found to be superior. Similarly these outcome vectors are also used to calculate the distinctions which determine whether any of the candidate tests should remain in the population, Step 2(c)vi.

The normalization of outcome vectors was done to prevent over-generalization in the form of persistent over-bidders. As explained below, a learner could

achieve maximum objective values on tests matching its action by always bidding as high as possible; the normalization factor is meant to penalize such individuals. At the same time, it does not penalize individuals for under-bidding emphasizing low bids on out-of-class instances.

Specific algorithm steps are detailed below.

Learner generation, Step 2(c)i. Each learner $l_i \in L_{pop}$ is considered in turn. With probability ρ a new learner is created from l_i through a series of search operators (Section 3.2), otherwise, a random learner is created. The new candidate learners are added to the population doubling its size.

Test generation, Step 2(c)ii. Test points index the training data and are generated without assuming any particular bias. Candidate tests are therefore generated by uniform sampling of the training dataset and eliminating duplicates until the size of T_{pop} is doubled.

Bid, Step 2(c)iiiA. The input test t_k is presented to each learner l_i which executes its bid program on t_k and outputs a bid $b \in (0, 1)$. The outcome of applying l_i to t_k is then calculated as

$$G(l_i, t_k) = \begin{cases} b & \text{if action of } l_i \text{ matches label of } t_k \\ 1 - b & \text{otherwise} \end{cases} \tag{4}$$

and appended to an initially empty outcome vector $\vec{o}_{l_i}$. Following the bidding round, each learner l_i has an associated outcome vector, Eq. 2.

Outcome normalization, Step 2(c)iv. For each outcome vector $\vec{o}_{l_i}$ a scaling factor μ_i is calculated as

$$\mu_i = \frac{1 + \Sigma_{t_k \in M} \vec{o}_{l_i}[k]}{1 + |M|} \tag{5}$$

where M is the set of tests in T_{pop} whose class does *not* match the action of l_i. Since each outcome falls in $(0, 1)$ the scaling factor μ_i is also limited to the unit interval. The normalized outcome vector is then calculated as $\vec{o}_{l_i} \leftarrow \mu_i \cdot \vec{o}_{l_i}$.

Learner selection, Step 2(c)v. The learner population L_{pop} is partitioned into the set of candidate learners C and the set of non-candidate learners P. Each learner $l_i \in C$ is then compared to each remaining learner $l_j \in P$. If $\vec{o}_{l_i}$ is found to dominate $\vec{o}_{l_j}$ according to Eq. 1 then l_j is removed from P, l_i is marked for selection, and the next learner in C is considered. The order in which the learners in C are considered is randomized, and for each learner in C that is considered, the order in which the learners in P are considered is randomized. Once all the learners in C are processed, L_{pop} in the next epoch is formed by combining the remaining learners in P with the learners in C marked for selection.

A candidate learner is considered for selection only if no candidate having the same outcome vector has already been seen in C in the current epoch. Also, a candidate vector is eliminated from consideration if a learner with the same outcome vector is observed in P. This process reduces the size of L_{pop} to what it was after initialization, Step 2a.

Test selection, Step 2(c)vi. Test selection is based on distinction vectors and follows exactly the same process (i.e., the way in which candidates are compared to non-candidates) as learner selection but with the following differences. First, checks for duplicates are not made. Second, the process makes sure that each instance class will be represented by at least one pattern in T_{pop} in the next epoch.

Reduction, Step 2d. Reduction is done after training for the current class is complete and the final set of learners and tests is selected. The outcome vectors for the learners are therefore updated to reflect the remaining tests in T_{pop}. Based on these updated outcome vectors any dominated or duplicate learners are then discarded from L_{pop}. Given that T_{pop} is coevolved to accurately distinguish between the learners, this step is meant to identify and remove redundant individuals.

Whenever outcomes were compared the equality operator was deliberately made imprecise. Two outcomes would be considered equal if their absolute difference was less than δ for relatively high values of δ such as 0.1. This was done to reduce the number of ways in which outcome vectors could vary and prevent situations where two learners were both selected despite their behaviour being virtually the same.

3.2 Linear Genetic Programming

Linear GP [5] was used to evolve the learners' bid procedures. Each program consisted of a sequence of binary instructions representing one- and two-operand operations applied to inputs and a set of null-initialized registers. Following program execution, the real-value GP output y was extracted from a predefined register. To obtain bid values in the unit interval, the Sigmoid function $f(y) = (1 - e^{-y})^{-1}$ was applied.

Four stochastic search operators were applied to the bid program of an existing learner to generate a candidate (training Step 2(c)i): (1) *delete* removed an arbitrary instruction, (2) *add* inserted a random instruction at an arbitrary location, (3) *mutate* flipped an arbitrarily selected bit in the program, and (4) *swap* exchanged the location of two arbitrary instructions. These operators were applied independently with a predefined probability. In all cases, a uniform distribution was used to select bits/instructions and to set bit values during the generation of random individuals.

Whenever a random learner was generated (training Steps 2a and 2(c)i), its bid program size was selected from a predefined range with uniform probability. The delete and add operators were included to allow varying program complexity within this fixed-length representation. The swap operator was added to remedy situations where the correct instructions were present but in the wrong order, and the mutate operator was included to alter a single field of an existing instruction.

Table 1: Summary of the datasets used in the evaluation.

	features	class exemplar counts		
		class	train	test
THY	21	1	93	73
		2	191	177
		3	3488	3178
		all	3772	3428
CEN	41	1	187141	93576
		2	12382	6186
		all	199523	99762
KDD	41	1	97277	60593
		2	1126	16347
		3	391458	229853
		4	4107	4166
		5	52	70
		all	494020	312029

4 Evaluation and Results

4.1 Datasets and Parameterization

The approach was evaluated on three large datasets. The Thyroid Disease (THY) dataset was obtained from the UCI Machine Learning Repository [1] while the Census Income (CEN) and KDD Cup 1999 (KDD) were found in the UCI KDD Archive [2]. Training on THY and CEN used the original training partitions leaving the remaining patterns for testing, while on KDD training was done using the '10% subset' dataset and testing was done on the 'corrected' dataset. The only preprocessing performed was to enumerate nominal attributes so they form valid GP inputs. The motivation for selecting these datasets was their large size (all), their multiple (versus two) classes (THY, KDD), and because of their unbalanced class distributions (all). These properties are summarized in Table 1.

The parameters used in evaluating the proposed approach are shown in Table 2. Here, the population sizes refer to the counts before generation, Steps 2(c)i and 2(c)ii of the training algorithm[3]. Since GP training is a stochastic process, thirty trials of each experiment were performed using different initializations.

To provide a baseline level of performance and to determine if using distinctions to select tests is effective, runs were also performed with a modified version of the algorithm. Here, Steps 2b and 2(c)vi of the original algorithm were discarded and test generation, Step 2(c)ii, was modified to fill the entire test population completely at random (i.e., T_{pop} would consist of one-hundred unique tests selected with uniform probability). Because the test population

[3]On THY, CEN, and KDD this corresponds to training using learner populations sizes of 60, 40, and 100 respectively using traditional, single-population, GP.

Table 2: Parameter values used in the experiments.

parameter	value
minimum program size	1
maximum program size	48
delete/add/mutate/swap prob.	0.5
number of registers	8
function set	$\{cos, sin, exp, log, +, \times, -, \div\}$
L_{pop} size	10
T_{pop} size	50
ρ	0.9
number of epochs	50 000
δ	0.1
number of initializations	30

was no longer coevolved to accurately evaluate the learners, the reduction step was also omitted.

Results were compiled on a class-by-class basis with respect to accuracy (ACC), detection rate (DET), and false positive rate (FPR). These were defined as

$$ACC = \frac{TN + TP}{TN + TP + FN + FP} \tag{6}$$

$$DET = \frac{TP}{FN + TP} \tag{7}$$

$$FPR = \frac{FP}{TN + FP} \tag{8}$$

where TN, TP, FN, and FP refer to true-negatives, true-positives, false-negatives, and false-positives respectively.

4.2 Results

4.2.1 Distinction-Based Test Point Population

Quartile accuracies using the proposed approach on the test data are shown in Table 3. On THY, class 3 accounts for nearly 92.5% of the training instances so it is not surprising to see the high DET values as well as the high FPR values. However, the approach manages to detect some of the minority classes and does surprisingly well on class 1 which accounts for just 2.5% of the training exemplars. On CEN the results also reflect the class imbalance with high DET and FPR values for class 1. On KDD despite 79.2% of training patterns falling in class 3, on this class the approach achieves very low FPR values.

Figures 2 and 3 show the bidding behaviour evolved using the proposed approach on THY and CEN respectively. The x-axis corresponds to cases (e.g.,

Table 3: Experimental results on the test data using the proposed approach. For each initialization, the ACC, DET, and FPR were calculated. Shown are the first quartile (Q1), median (MED), and third quartile (Q3) values over all initializations.

		class	Q1	MED	Q3
THY	ACC	1	0.978	0.982	0.986
		2	0.945	0.949	0.963
		3	0.942	0.952	0.961
	DET	1	0.592	0.692	0.890
		2	0.086	0.393	0.585
		3	0.973	0.980	0.987
	FPR	1	0.007	0.010	0.016
		2	0.006	0.015	0.019
		3	0.262	0.390	0.571
CEN	ACC	1	0.902	0.918	0.928
		2	0.902	0.918	0.928
	DET	1	0.936	0.956	0.975
		2	0.184	0.303	0.427
	FPR	1	0.573	0.697	0.816
		2	0.025	0.044	0.064
KDD	ACC	1	0.918	0.920	0.924
		2	0.947	0.947	0.948
		3	0.967	0.972	0.974
		4	0.991	0.992	0.993
		5	1.000	1.000	1.000
	DET	1	0.976	0.990	0.995
		2	0.000	0.010	0.035
		3	0.960	0.967	0.970
		4	0.531	0.628	0.715
		5	0.000	0.064	0.207
	FPR	1	0.090	0.095	0.099
		2	0.000	0.000	0.003
		3	0.003	0.006	0.015
		4	0.001	0.001	0.004
		5	0.000	0.000	0.000

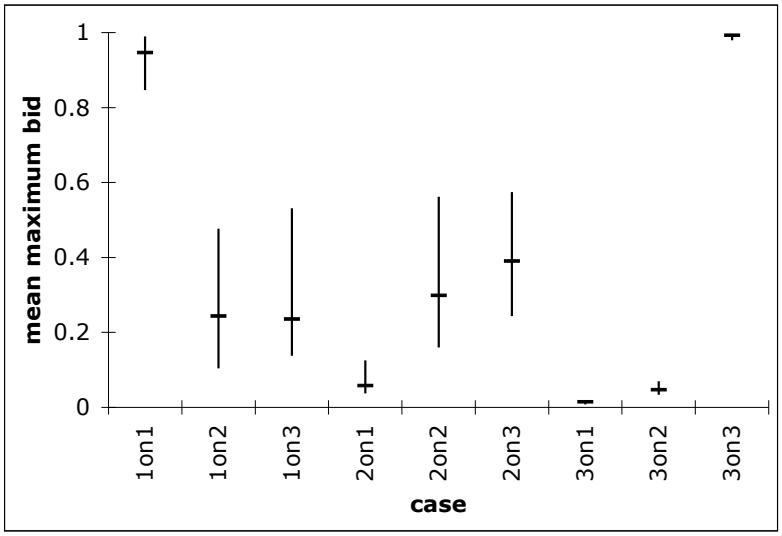

Figure 2: Bidding behaviour on THY test data. Compiled over all initializations, the vertical line endpoints denote the first and third quartiles and the vertical segment denotes the median.

learners of some action a bidding on tests of some class c) and the y-axis corresponds to the mean maximum bid for each case. Whenever an instance of class c is observed, the bids of all the learners of action a are considered and the maximum bid selected. The mean maximum bid for a single initialization is then calculated as the arithmetic mean of this maximum value over all observations. In short, ideal bidding behaviour would correspond to maximizing cases in which c equals a and minimizing all other combinations.

The bidding behaviour shown is consistent with the accuracy results from Table 3. Specifically, if the approach results in high DET values on a class c, then the bid values for the case '$c$on$c$' tend to be consistently high. For example, on THY, learners of action 3 almost always bid close to 1 on instances of class 3. On classes 1 and 2, these learners bid consistently low suggesting that classification error results from poor bidding behaviour by learners matching the minority classes. This is evident in Figure 2 where bids for case '1on2' overlap closely with bids for case '2on2'. The same behaviour is seen on the CEN and (not shown) KDD datasets.

Table 4 shows that on THY all classes tend to be represented in T_{pop} so that learners have the opportunity to recognize out-of-class exemplars. However, most of the instances in T_{pop} always correspond to the majority class in the underlying dataset, class 3. In addition, the counts rise for an instance class whenever it matches the action of the learners being evolved. For example, class 1 counts are highest when learners of action 1 are evolved. Thus, it appears the

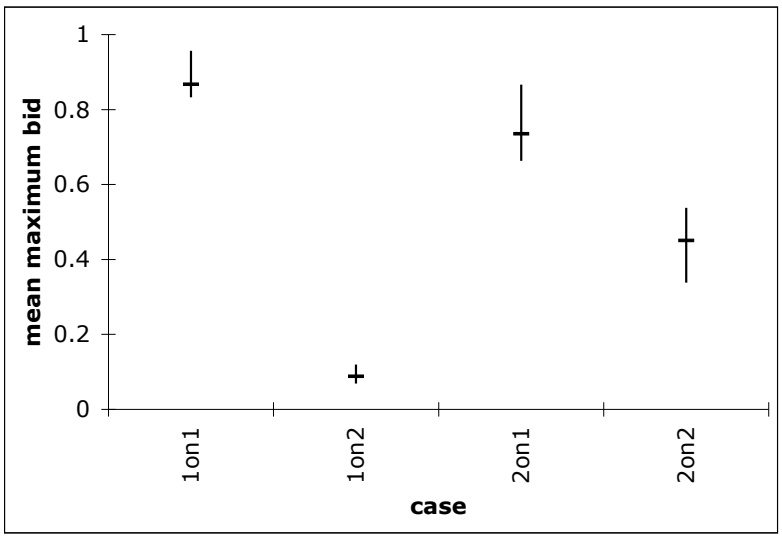

Figure 3: Bidding behaviour (analogous to Figure 2) on CEN test data.

coevolution seeks out in-class exemplars as well.

Table 5 shows that within each exemplar class the approach decomposes the instance space. For example, the median number of learners responsible for identifying THY exemplars of class 1 is seven. What is not clear given the current data is the form of this decomposition: do all seven learners correctly identify roughly the same fraction of class 1 exemplars (and raise the same number of false positives), or is there a learner that identifies the majority of class 1 exemplars leaving the others with minimal contributions? Regardless, the approach automatically determines how many individuals need to participate in the final solution while at the same time evolving the behaviour that defines how these individuals are to be combined to form that solution.

Finally, training times on the three datasets are shown in Table 6. They were collected using the C *getrusage*() procedure and account for both system and user time. Although there is room for improvement (runs on KDD using PGPC took minutes) these run times are significantly faster that GP without coevolution [12] (e.g., on the KDD dataset a single run on one class using canonical GP takes eleven hours to complete).

4.2.2 Random Test Point Population

Results using the baseline approach are shown in Table 7. On CEN, the baseline approach does worse producing much higher FPR values on class 1 and much lower DET values on class 2. On KDD, the results on class 1 and class 3 are similar to the values produced by the proposed approach. However, on class

Table 4: Number of tests of each class in T_{pop} at the end of training on THY corresponding to the runs in Table 3. The counts are grouped based on the action of the evolved learners. Shown are the first quartile (Q1), median (MED), and third quartile (Q3) values over all initializations.

action	class	Q1	MED	Q3
1	1	6.25	12.00	20.75
	2	3.00	6.00	10.00
	3	20.50	27.50	40.75
2	1	1.00	3.00	5.75
	2	11.00	16.50	20.00
	3	25.00	28.50	37.00
3	1	1.00	1.50	3.00
	2	6.25	10.00	15.00
	3	29.50	36.00	42.75

Table 5: Number of learners in the final solution corresponding to the runs in Table 3. Shown are the first quartile (Q1), median (MED), and third quartile (Q3) values over all initializations.

	class	Q1	MED	Q3
THY	1	6.00	7.00	9.00
	2	5.25	8.00	10.00
	3	6.25	7.00	8.00
CEN	1	6.00	8.00	10.00
	2	7.00	8.00	9.00
KDD	1	5.00	7.00	8.00
	2	1.25	5.50	6.75
	3	1.00	5.00	5.75
	4	1.00	3.00	5.00
	5	1.00	1.00	1.00

Table 6: Training times in minutes corresponding to the runs in Table 3. Shown are the first quartile (Q1), median (MED), and third quartile (Q3) values over all initializations.

	THY	CEN	KDD
Q1	50	65	151
MED	52	67	154
Q3	52	68	156

4, the distinction-based test point generation scheme results in a significant improvement in the detection rate without compromising the false positive rate. On THY, the baseline approach appears to outperform the proposed approach yielding higher DET values on class 1 and lower FPR values on class 3.

4.2.3 Other Classifier Results

In general, meaningful comparison with other approaches is difficult because the aggregate accuracy values that are typically reported hide class-wise rates that are significant whenever datasets are unbalanced. This was not the case on KDD where the winning entry in the KDD Cup achieved DET rates of 99.5%, 8.4%, 97.1%, 83.3%, and 13.2% on classes 1, 2, 3, 4, and 5 respectively [7]. The proposed approach provides competitive rates of 99.5%, 3.5%, 97.0%, 71.5%, and 20.7% on the respective classes. On THY, classifiers based on accuracy [3] were able to achieve classification rates of 86.9% on the test data while the mean accuracy of the proposed approach over all initializations was 94.1%. The mean test accuracy on CEN using the proposed approach was 90.3% which is less than the reported accuracy of 95.3% using C5.0 [1]. This comparison, however, does not include DET rates, which are important given the unbalanced nature of this dataset (labeling all exemplars as the major class would provide accuracy of about 95%).

4.3 Additional Experiments Results

An unexpected outcome from the first set of experiments was that the baseline approach appeared to outperform the proposed approach on THY. Initial speculation was that the proposed approach overfit the training data but analysis of the training data accuracies relative to the test data accuracies (not shown) did not suggest this. To determine if perhaps the size of L_{pop} was insufficient, a set of experiments was performed where the size of the learner population was doubled. Although the performance of the original approach improved significantly, the performance of the baseline approach improved more dramatically, Table 8.

A possible explanation for this unexpected outcome is as follows. A learner that identifies just a single instance of a matching class (e.g., wins the bidding round for this instance but otherwise bids low) may persist in L_{pop} if no other learner does well on that instance. Since a limited set of learners is allowed to persist across epochs and because the criterion for entry into this set is Pareto-dominance of a non-candidate, a candidate learner may do well on many other instances and so be more desirable yet still be discarded. Thus, L_{pop} may be evolved to contain learners that identify just a handful of instances. Similarly, in T_{pop}, the tests which these specialized learners identify will tend to persist in a symbiotic relationship with the learners because they make the distinctions between the specialist and the non-specialists. The problem is compounded by the fact that new candidates are generated from L_{pop} after selection limiting the exploration of the solution space. With the baseline approach, these specialists

Table 7: Experimental results (analogous to Table 3) on the test data using the baseline approach.

		class	Q1	MED	Q3
THY	ACC	1	0.971	0.980	0.989
		2	0.950	0.955	0.962
		3	0.953	0.961	0.973
	DET	1	0.777	0.904	0.932
		2	0.071	0.319	0.712
		3	0.980	0.985	0.989
	FPR	1	0.008	0.016	0.027
		2	0.001	0.006	0.016
		3	0.143	0.284	0.453
CEN	ACC	1	0.934	0.938	0.940
		2	0.934	0.938	0.940
	DET	1	0.986	0.990	0.994
		2	0.124	0.152	0.168
	FPR	1	0.832	0.848	0.876
		2	0.006	0.010	0.014
KDD	ACC	1	0.920	0.923	0.925
		2	0.945	0.947	0.948
		3	0.947	0.966	0.971
		4	0.986	0.987	0.991
		5	0.999	0.999	1.000
	DET	1	0.971	0.983	0.988
		2	0.023	0.033	0.036
		3	0.937	0.967	0.969
		4	0.157	0.264	0.547
		5	0.086	0.143	0.275
	FPR	1	0.087	0.091	0.095
		2	0.001	0.002	0.005
		3	0.018	0.031	0.038
		4	0.001	0.003	0.005
		5	0.000	0.000	0.001

Table 8: Experimental results (analogous to Table 3) on the test data using the baseline and original approaches using twice as many learners in L_{pop}.

		class	Q1	MED	Q3
THY (original)	ACC	1	0.973	0.979	0.987
		2	0.945	0.952	0.960
		3	0.934	0.952	0.960
	DET	1	0.565	0.719	0.836
		2	0.346	0.647	0.798
		3	0.946	0.968	0.983
	FPR	1	0.004	0.010	0.024
		2	0.010	0.025	0.046
		3	0.121	0.230	0.422
THY (baseline)	ACC	1	0.972	0.986	0.989
		2	0.954	0.977	0.983
		3	0.965	0.974	0.978
	DET	1	0.818	0.890	0.945
		2	0.179	0.771	0.903
		3	0.981	0.983	0.988
	FPR	1	0.008	0.012	0.027
		2	0.003	0.010	0.013
		3	0.048	0.110	0.248

are less likely to persist as it is unlikely that their counterpart tests will continue to be generated every epoch.

This conjecture is supported by the learner turnover rates, Figure 4, or the number of times a non-candidate is replaced by a candidate at different stages of training for each initialization of the additional runs. The turnover rates for the original approach (on average, one candidate learner was accepted every seventeen epochs) are much lower than for the baseline approach indicating that learners persist for much longer. However, neither trend line plateaus and the constant linear increase suggests that learning and forgetting is taking place; this would also account for the improved results achieved by doubling the size of L_{pop} (i.e., increasing the memory size). Results for the other actions were similar.

Despite the fact that neither of the trendlines in Figure 4 plateaus, the system manages to reach a stable level of performance, Figures 5 and 6. Figure 5 shows the sum over all dimensions of the (unnormalized) outcome vectors of the 20 learners remaining in L_{pop} with respect to the 50 tests remaining in T_{pop} after selection (maximum value 1000). Figure 6 is similar except that the sum is taken over all 100 tests in the population (since there is no selection) and then halved to make comparison easier. Interestingly, despite selecting an entirely new T_{pop} every epoch the degree of variance in the baseline approach is lower. In addition, performance reaches a plateau around epoch 10000 suggesting that training for 50000 epochs may be far too long. Results for the other two actions

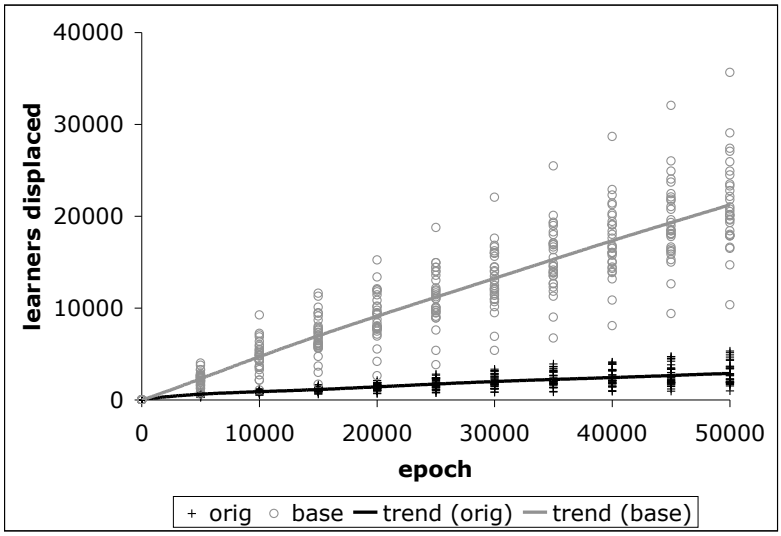

Figure 4: Turnover rates on THY for learners of action 1. The cumulative number of displaced non-candidate learners is plotted every 5000 epochs for each initialization using the original (orig) and baseline (base) approaches.

were also similar.

The failure of learners to generalize as suggested above has been identified in coevolutionary systems as *focusing* while the learning and forgetting results from *relativism* [15]. Given the complexity of the problem domain in this work further analysis is required to identify the exact cause of the observed behaviour and how it relates to these ideas.

5 Conclusions

In this work, a bid-based approach to classification was presented in which individuals evolve bidding behaviour for a preassigned action (class label). The proposed algorithm coevolves a population of GP-based learners alongside a population of training instances. The learners attempt to decompose the problem by way of their bidding behaviour while the goal of the tests is to distinguish between the learners. Since the test population represents a small subset of all training instances, the algorithm scales to large datasets (i.e., the cost of fitness evaluation is decoupled from the size of the original dataset). Problem decomposition appears as a natural property of the system as individuals are only associated with exemplars on which they provide the winning bid. Thus, individuals cooperate to decompose the problem while competition ensues between learners and the subset of tests. Class-wise measures of performance were

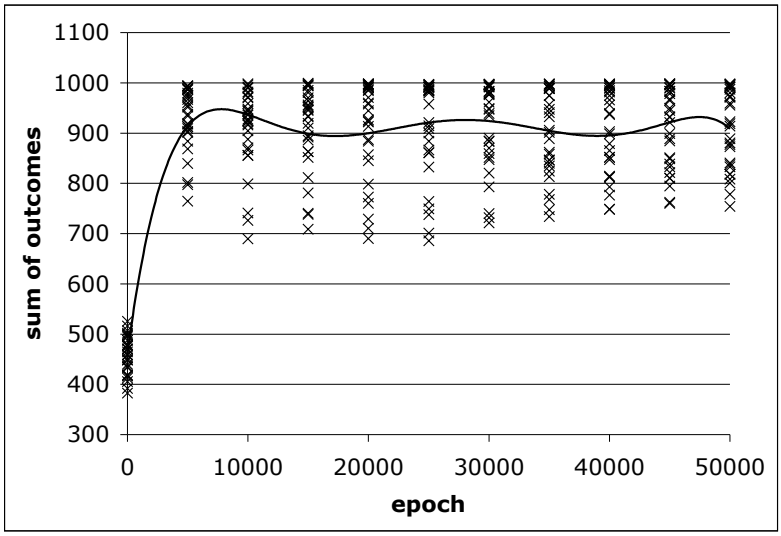

Figure 5: Learning curve on THY for learners of action 1 using the original approach. The sum of the outcome vectors of the surviving learners with respect to the surviving tests is plotted for each initialization along with a trendline.

reported and the approach was found to be competitive on large, multi-class problems using much reduced computational overhead.

Due to the strict criterion for acceptance of candidate learners and tests, the algorithm was suspected focusing on a small number of exemplars resulting in individuals that generalized poorly. Additional experiments need to be performed to analyze the populations and determine the exact coevolutionary dynamics that occur. Since scalability is a goal of the proposed approach, early stopping criteria such as the stagnation of the Pareto front should also be investigated. Finally, a thorough analysis of how the learners decompose the instance space would provide useful insight into the behaviour of the generated models.

6 Acknowledgments

This work was conducted while Peter Lichodzijewski was supported by an NSERC PGS-D Scholarship and a Killam Postgraduate Scholarship. Malcolm. I. Heywood would like to thank NSERC, MITACS, and CFI for their financial support.

References

[1] D. J. Newman and S. Hettich and C. L. Blake and C. J. Merz. UCI Repository of Machine

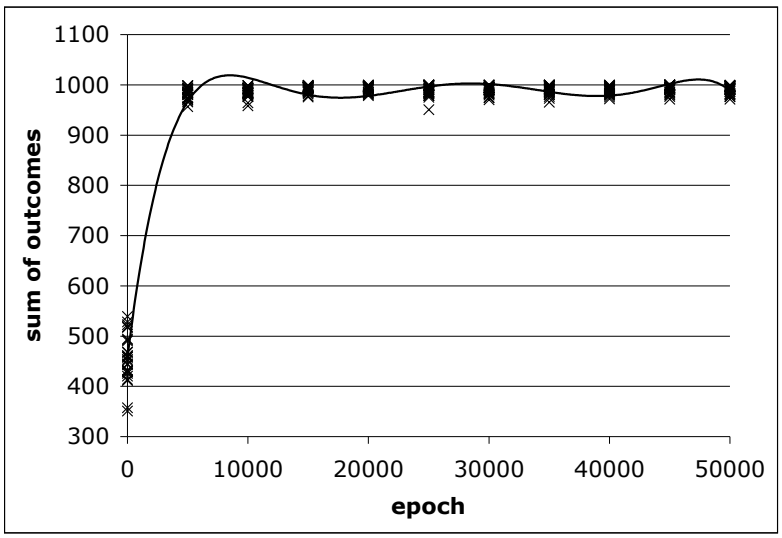

Figure 6: Learning curve (analogous to Figure 5) on THY for learners of action 1 using the baseline approach.

Learning Databases [http://www.ics.uci.edu/∼mlearn/mlrepository.html]. Irvine, CA: University of California, Dept. of Information and Comp. Science, 1998.

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

[3] E. Bernado-Mansilla and J. M. Garrell-Guiu. Accuracy-based learning classifier systems: Models, analysis and applications to classification tasks. *Evolutionary Computation*, 11(3):209–238, 2003.

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

[5] M. Brameier and W. Banzhaf. *Linear Genetic Programming*. Springer, Genetic and Evolutionary Computation Series, 2007.

[6] E. D. De Jong and J. B. Pollack. Ideal evaluation from coevolution. *Evolutionary Computation*, 12:159–192, 2004.

[7] C. Elkan. Results of the KDD'99 classifier learning. *SIGKDD Explorations*, 1(2):63–64, 2000.

[8] S. G. Ficici and J. B. Pollack. Pareto optimality in coevolutionary learning. In *Proceedings of the 6th European Conference on Advances in Artificial Life*, pages 316–325, 2001.

[9] G. Folino, C. Pizzuti, and G. Spezzano. Boosting technique for combining cellular GP classifiers. In *Proceedings of the European Conference on Genetic Programming*, pages 47–67, 2004.

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

[11] W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002.

[12] M. Lemczyk and M. I. Heywood. Training binary GP classifiers efficiently: A pareto-coevolutionary approach. In *Proceedings of the European Conference on Genetic Programming*, pages 299–240, 2007.

[13] A. R. McIntyre and M. I. Heywood. MOGE: GP classification problem decomposition using multi-objective optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 863–870, 2006.

[14] J. Noble and R. A. Watson. Pareto coevolution: Using performance against coevolved opponents in a game as dimensions for pareto selection. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 493–500, 2001.

[15] R. A. Watson and J. B. Pollack. Coevolutionary dynamics in a minimal substrate. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 702–709, 2001.

[16] S. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.