Training Binary GP Classifiers Efficiently: a Pareto-coevolutionary Approach*

Michal Lemczyk and Malcolm I. Heywood†

July 17, 2007

Abstract

The conversion and extension of the Incremental Pareto-Coevolution Archive algorithm (IPCA) into the domain of Genetic Programming classification is presented. In particular, the coevolutionary aspect of the IPCA algorithm is utilized to simultaneously evolve a subset of the training data that provides distinctions between candidate classifiers. Empirical results indicate that such a scheme significantly reduces the computational overhead of fitness evaluation on large binary classification data sets. Moreover, unlike the performance of GP classifiers trained using alternative subset selection algorithms, the proposed Pareto-coevolutionary approach is able to match or better the classification performance of GP trained over all training exemplars. Finally, problem decomposition appears as a natural consequence of assuming a Pareto model for coevolution. In order to make use of this property a voting scheme is used to integrate the results of all classifiers from the Pareto front, post training.

1 Introduction

Binary classification problems within the context of a supervised learning paradigm provide the basis for a wide range of application areas under machine learning. However, in order to provide scalable as well as accurate solutions, it must be possible to train classifiers efficiently. Although Genetic Programming (GP) has the potential to provide classifiers with many desirable properties, the computational overhead in doing so has typically been addressed through hardware related solutions alone [9],[2],[5]. In this work we concentrate on how the training process can be made more efficient by evaluating classifier fitness over some adaptive subset of the total training data. To date, the typical approach has been to utilize an active learning algorithm for this purpose, where the Dynamic Subset Selection (DSS) family represents one widely used approach [3],[8],[11].

*Published in EuroGP'07. LNCS 4445. Springer-Verlag

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

In this work, an alternative approach to the problem is presented in which the problem is designed as a competition between two populations, one representing the classifiers, the other the data. Progress has recently been made using Genetic Algorithms based on a Pareto formulation of the competitive coevolutionary approach, albeit within the context of player behaviours in gaming environments. To this end, the proposed approach is based on the Incremental Pareto-Coevolution Archive (IPCA) algorithm, where this has been shown to address several potential problems with the competitive coevolutionary paradigm i.e., relativism, focusing, disengagement, and intransitivity [1].

The algorithm reported in this work, hereafter denoted the Pareto-coevolutionary GP Classifier (PGPC) is novel in the fact that it extends a Genetic Algorithm "game-playing" context into the domain of GP classification. Furthermore, pruning is utilized to limit the sizes of the IPCA algorithm archives – the point and learner pareto-fronts – to allow for efficient execution. This differs from the method employed in the follow-up of the IPCA algorithm, the Layered Pareto-Coevolutionary Archive (LAPCA) [4], which relies on storing the top N pareto-layers of the archive, keeping the pareto-front in its entirety. Additionally, PGPC differs from the methods utilized by various Evolutionary Multi-Objective Optimization (EMOO) algorithms, which tend to perform clustering on the pareto-front of solutions using the coordinates of candidate solutions to limit the size of the pareto-front [6], [12]. That is to say, the cooperative coevolutionary case of EMOO is able to maintain limits on the size of the Pareto front through similarity measures applied pairwise to candidate solutions. In a GP environment, the design of a suitable similarity metric is not straightforward as learners take the form of programs. As a consequence, this work investigates pruning heuristics that make use of structure inherent in the interaction between learners and training points. Thus, the learner archive is pruned relative to a performance heuristic defined over the contents of the point archive, and the point archive is pruned relative to a heuristic quantifying class distribution and point similarity.

In addition, the GP context requires an alternative approach from those employed previously when resolving which solution from the pareto-front to apply under post training conditions. Specifically, an EMOO context does not face this problem as a individual is identified from the pareto-front of solutions on the basis of a distance calculation. The solution with minimum distance relative to the unseen test condition represents the optimal response. Conversely, under a GP classification context all individuals from the pareto-front provide a label i.e., only under training conditions are we able to explicitly identify which classifier is optimal through the associated classification error. Thus, instead of a single individual representing the optimal strategy for each exemplar, a voting policy is adopted in which all members of the pareto-front provide labels for each exemplar under post training conditions.

1.1 Approach

The co-evolutionary approach of the IPCA algorithm will allow for the "binding" of the learner and training data point subset evolutions, keeping the point subset relevant to the current set of learners. The pareto-front of learners allows the system to explore the search space along different attractors present in the data, and hopefully provide a diverse set of optimal solutions. In regards to the pareto-front of points, each point is pareto-equivalent to the others in the front, and as such provides a "distinction" between the learners that is not duplicated in the archive. Therefore the pareto-front of points itself is the subset of training data that provides a learning gradient for the learners [1], [4].

The original IPCA algorithm performed no pruning on the learner and point archives. Empirically, the learner archive (pareto-front) remained small, and the point archive which contained the current set of relevant points in addition to the previously relevant set, grew without bounds [1]. In the case of the proposed PGPC algorithm, experiments showed that both the learner and point pareto-fronts grow dramatically on the training data sets, since it may be that each training point is an underlying objective and provides a distinction between learners. To retain efficiency, the previously relevant points may not be stored, nor can the pareto-fronts in their entirety. A pruning method must be adopted to limit the size of the archives.

Furthermore, in the context of a classification problem, a heuristic is required to define how the pareto-front of learners is consolidated to return one classifier per testing point. Since the pareto-front of learners may be diversified to correctly classify subsets of the data, a method to recognize and utilize any structure inherent in the front must be developed such that the most appropriate classifier responds under unseen data conditions. This is generally not a problem within the context of EMOO as solutions take the form of a point in a coordinate space. Identifying which individual represent the best solution is resolved through the application of a suitable distance metric. Under the GP (classification) domain, solutions take the form of models providing a mapping from a multi-dimensional input space to a 1-dimensional binary output space. Thus, on unseen data it is not possible to associate exemplars to models a priori. This problem is addressed in this work by adopting a simple voting scheme over the contents of the learner archive.

2 The Pareto-coevolutionary GP Classifier Algorithm

In terms of the GP individuals or learners, a tree-structured representation is employed, whereas individuals in the point population(s) index the training data. The classical GP approach for interpreting the numerical GP output ($gpOut$) in terms of a class label is utilized, or

IF ($gpOut \leq 0.0$) THEN (return class 0), ELSE (return class 1) (1)

The PGPC algorithm utilizes four populations of individuals: (1) a fixed size learner population which provides the exploratory aspect of the learner evolution. (2) a learner archive which contains the pareto-front of learners, bound by a maximum size value. (3) a fixed size point population (point population << training exemplar count). (4) a point archive which describes the current subset of training points relevant to the learner archive, bound by a maximum size value. Figure 1 summarizes the organization of data dependencies for each step of the algorithm.

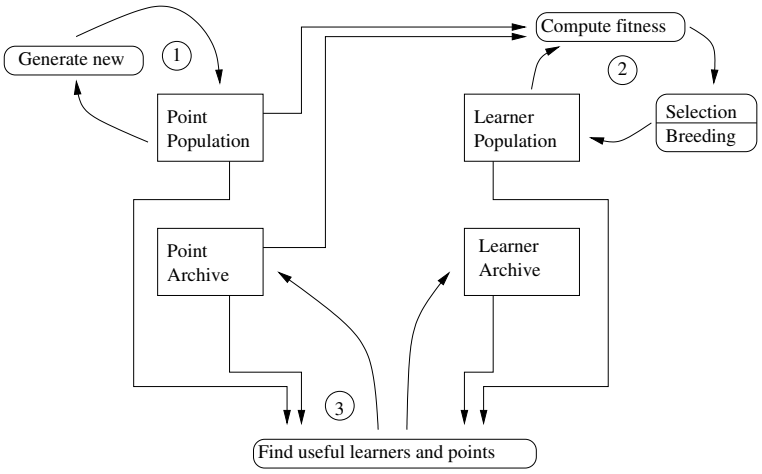


Figure 1: The PGPC architecture.

The PGPC algorithm consists of the following steps performed at each generation of evolution:

Step 1: Generate points in the point population: Since the points are indices within the training data, a crossover operator makes little sense. Therefore, only mutation was utilized to generate the point population, with mutation being performed on each population member. Furthermore, to ensure class balance within the point population, each half of the population is randomly filled with points belonging to the same class, Figure 1 point 1.

Step 2: Generate learners in the learner population: The canonical tree-structured model of GP is assumed [7], although the PGPC algorithm could utilize any standard GP model for the representation and definition of search and selection operators. In this case fitness proportionate selection is

assumed, with fitness being calculated over the contents of both the point population and point archive, Figure 1 point 2.

Step 3: The following steps deal with the entry criteria for the point and learner archives, Figure 1 point 3:

3.a: Compute the set of useful points regarding the learner population and archive: As per IPCA; if a newly generated learner is dominated by the learner archive or contains equal values (evaluated over the point archive), and the addition of a new point into the evaluation set provides a distinction such that the generated learner is pareto-equivalent to the archive with no equal values, the point is inserted into the archive.

3.b: Compute the set of useful learners regarding the point population and archive: As per IPCA; any generated learner that is pareto-equivalent to the archive with no equal values (again, evaluated over the point archive) enters the archive. Furthermore, if a generated learner is non-dominated, and a generated point defeats it, they both enter their respective archives.

3.c: Remove duplicates in the learner and point archives and newly-dominated learners in the learner archive: As per IPCA.

To maintain the efficiency of the algorithm, a limit on the archive sizes is enforced. This limit may be thought of as a tunable parameter of efficiency vs accuracy. However the relationship between the two may depend on the classification problem, as the number of underlying objectives which the point archive strives to evolve towards may vary. Moreover, the number of individuals qualifying for inclusion in the point or learner archives is also a dynamic property, with a significant difference between the number qualifying in early versus later generations.

Within the context of the pruning algorithm, the first assertion will be that a newly generated learner or point should enter the archive at the possible cost of evicting an older member. This assertion will help avoid stagnation within the archive at the risk of possible regression or forgetting. Alternate insertion basis may be considered and evaluated in the future. Within this framework, all that remains is to provide a basis for selection of an archive individual for replacement.

For pruning the learner archive, the proposed greedy approach consists of removing the learner with the worst performance against the point archive (with the measure being the number of incorrectly classified instances). Within this view, a learner to be removed may have entered the archive by simply correctly classifying one training archive point while misclassifying the remainder, therefore removing the "worst" learner deletes some of the explorative diversity of the archive in favour of increased average accuracy.

For pruning the point archive, the proposed basis utilizes the genotypic information of the point co-ordinates to delete one of the two closest points, distance defined using the Euclidean metric, adhering to the following criteria: the two points must be of the same class, and that class must be over-represented in the point archive. This approach will promote class-balanced diversity in the point archive, while preserving the points which define boundaries between clusters of points.

Finally, in order to attain a measure of classification performance on testing data at the completion of training, the learner pareto-front must be interpreted to provide one class prediction per testing point. To this end an "Average archive value" voting scheme is used in which each pareto-front learner provides one vote for their class prediction of the input testing point. The class with the majority of the votes is selected as the system's prediction for the corresponding data point. Such a scheme is adopted to make use of the aforementioned learner pruning bias in which learners are rewarded for maximizing the number of correctly classified exemplars. Thus, we expect classification by consensus as opposed to outright specialization, in which case a more sophisticated voting policy might be required.

3 Experiments

Evaluation of the proposed PGPC algorithm will be performed against canonical tree-structured GP and three alternative active learning algorithms. Indeed all methods will share the same canonical GP model. The three active learning comparison algorithms use a limit on the number of training exemplars to provide a more accurate comparison with the PGPC classifier. These algorithms will allow for an evaluation of the IPCA-based dynamics and solution space search efficiency. They differ from the canonical GP algorithm only in the active learning algorithm used to identify the subset of training exemplars over which fitness evaluation takes place. Section 3.2 summarizes the properties of each alternative algorithm.

Post training evaluation of classification performance is conducted using a single classification "score". This is based on a combined equal weighting of detection and false positive rate, or

$$Score = \frac{\frac{TruePositives}{Positives} + (1 - \frac{FalsePositives}{Negatives})}{2} \tag{2}$$

Adoption of such a metric will establish how robust alternative schemes are to any under representation of the minor class. Algorithm efficiency is measured in terms of run-time on a common machine under the same conditions.

The classification data sets used in the experiments consist of: Adult, and

KDD99[1]. Each data set is considered a two-class problem, with the training partition for Adult being set at 75%. The Adult data set consists of 33916 training points, and 11306 testing points (i.e., exemplars with missing features are not included); each having a dimension of 14 features. The KDD99 set consists of 494020 and 311027 training and testing points, respectively; with each point having a dimension of 41 features. In order to cast the problem as a binary classification problem we concentrate on separating the class representing 'normal' from the other four classes. Both data sets are unbalanced with approximately 20 percent in-class exemplars in KDD99 and 15 percent in-class exemplars in Adult.

The relevant hardware of the test machine utilized for the run-time experiments consists of: Pentium 4, 2.60GHz HT, 800MHz FSB. 1GB DDR400 RAM. 36GB SATA 10K-RPM Hard Drive. The GP implementation common to all five methods benchmarked was based on the lilGP[2] framework, running on Fedora Core 3 Linux.

3.1 Parameters

The tree-structured GP parameters common to all of the algorithms are summarized in Table 1. The sizes of the populations and archives of the PGPC algorithm will all be set to a common value of 25.

Although the learner archive is the set of learners constituting an "answer", the learner population is still used for exploration and the evolution of the learners. Therefore the sum of both the archive and population sizes is used to provide an equivalent limit on the number of learners utilized by the comparison algorithms, Table 2. The same holds true for the point population and archive, since they are both used in the evaluation of the fitness of a learner, they constitute the number of points that the algorithm can "access". Therefore the comparison subset selection algorithms utilize a subset size set to be the sum of the two, Table 2.

Due to the stochastic nature of the GP based algorithms, performance is reported over a total of 30 different population initializations per experiment.

3.2 Comparison Algorithms

Canonical tree-structured GP: The base line comparison algorithm takes the form of a canonical tree-structured GP classifier [7] (denoted as "Regular"), consisting of only one learner population. At every generation, the fitness of each

[1]Available at:
http://www.ics.uci.edu/∼mlearn/MLSummary.html [Adult]
http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html [KDD99]

[2]http://garage.cps.msu.edu/software/lil-gp

Table 1: GP parameters common to all of the algorithms.

Number of generations	500
Individual initialization method	Half and half
Individual initialization depth	2-6
Individual maximum nodes	1000
Individual maximum depth	17
Learner breeding phase selection method	Fitness Proportionate
Learner breeding phase operators	Crossover, Mutation
Learner breeding phase operator frequencies	0.8, 0.2
Function Set	*, /, +, -, sin, cos, exp, sqrt

Table 2: Population and Point Subset sizes for comparison algorithms.

Model	Regular	Cycling	Random	DSS
Learner Population Size	50	50	50	50
Point Subset Size	as per original Training Set	50	50	50

learner is computed using the entire training data set. The absolute switching function wrapper maps the *gpOut* value of the individual against the training data point class, equation (1), and the number of correct mappings (classifications) is recorded and normalized into a fitness value (accuracy). The fitness values of the individuals are used to perform fitness proportionate selection for breeding the next generation of individuals. Upon completion of the evolution, the fittest individual is used to classify the testing data using the same switching function[3].

Dynamic subset selection (DSS): This comparison algorithm is an implementation of the DSS algorithm [3]. Each training exemplar has an associated age and difficulty value; corresponding to the number of generations elapsed since that exemplar was last utilized in the subset, and the number of correct classifications of it when it was utilized. At each generation, a weighing of the two values is performed to yield a selection probability, or

$$PointWeight_p = Difficulty_p^{1.0} + Age_p^{3.5} \tag{3}$$

The selected points define a subset over which learners are evaluated in the current generation.

Random subset selection: This algorithm is based on the "Stochastic

[3]The score metric of equation (2) is only employed in the post training evaluation of performance.

sampling" method described in [10]. Specifically, individuals are evaluated over a subset of training data points selected randomly with uniform probability and used to compute the fitness of the individual.

Cycling subset selection: At any evaluation of any individual, a global index into the training data is incremented. Training points subsequent to the index are utilized to compute the fitness. Wrap-around is used to resolve the special case associated with the end of the training data set.

4 Results

Figure 2 illustrates test set performance using first quartile, median, and third quartile under the 'score' metric of equation (2). It is apparent that the PGPC algorithm matched or bettered all of the comparison algorithms, including the Regular GP algorithm. This indicates that the proposed algorithm has indeed provided an effective alternative mechanism for selecting subsets of training exemplars to perform the learner evolution upon. Moreover, although the fitness metric was based on classification count during training, it is also apparent that degenerate solutions were successfully avoided in the ensuing solutions (degenerates are equivalent to a score of 0.5). Conversely, all three alternative subset selection algorithms returned performance scores equivalent to degenerate solutions on the Adult data set. DSS performed better on KDD99, although never as good as PGPC and Regular GP; whereas both random and cycling subset selection performed worse on KDD99 than on the Adult data set. The consistently poor behavior of the Random sampling algorithm on KDD99 was associated with the population being dominated by degenerate solutions that labeled all exemplars as out of class, where this represents around 400,000 of the 500,000 training exemplars.

Table 3: Median scores and run-times in seconds of the various algorithms upon the Adult and KDD99 data sets.

Algorithm	Median score (Adult)	Median time (Adult)		Median score (KDD99)	Median time (KDD99)
PGPC	0.736611	41.38		0.918419	56.97
Regular	0.611569	1973.74		0.909291	40347.74
DSS	0.526903	11.29		0.827497	120.87
Random	0.527521	3.63		0.500000	4.20
Cycling	0.520470	3.46		0.501884	2.58

In terms of execution efficiency, Table 3, the PGPC algorithm exhibited a

speedup of 48 (Adult) to 708 (KDD99) with respect to Regular GP. The Random and Cycling Subset Selection algorithms were naturally the fastest, but also resulted in degenerate solutions for both data sets. The DSS algorithm was significantly faster than PGPC under Adult, however, was actually slower than PGPC under the larger KDD99 data set by a factor of two.

In summary, the PGPC algorithm was able to match (KDD) or better (Adult) the classification scores of the regular GP algorithm whilst providing a significant speedup. Moreover, as the size of the training data set increased, the computational effectiveness of the PGPC algorithm improves, Table 3. Thus, DSS takes twice as long to provide solutions on the KDD data set than PGPC, for no improvement in classification score. Conversely, the naive schemes for building subsets of exemplars, Random and Cycling, are very fast, but do not provide a useful model for learning, barely reaching a 50 percent classification score i.e., degenerate solutions were the norm.

5 Conclusion

An algorithm employing the coevolution of both classifiers and training data subset members within a Genetic Programming environment was presented. Comparisons were made to a traditional GP classifier employing the training data in its entirety, in addition to classifiers using only a subset of the data selected via either a Random, Cycling, or DSS method.

With regards to classification performance, the PGPC algorithm out-performed each of the comparison algorithms, indicating that the algorithm does not compromise classification performance. We conclude that even with a small subset of training points, the pareto-evolutionary approach to learner and point coevolution may generate superior or equivalent classification performance. Moreover, the PGPC algorithm was particularly effective at avoiding degenerate solutions; where this is particularly useful on problems described by large unbalanced data sets.

In the case of training efficiency, the training point subset selection of PGPC provides a dramatic increase in execution speed, without recourse to specialist hardware. Moreover, the approach becomes increasingly effective as the number of exemplars in the data set increases. Finally, we note that as the entire Pareto front of learners takes part in the solution, problem decomposition is supported as an additional side effect of adopting a coevolutionary paradigm.

Future work will evaluate the significance of different point and learner archive pruning schemes, as well as qualify the significance of pruning in practice. The latter is particularly relevant with regards to the impact of 'regression' or 'forgetting' on the behavior of the point and learner archives.

6 Acknowledgments

The authors gratefully acknowledge the support of CFI New Opportunities, NSERC Discovery, and MITACS grants (Canadian government), and SwissCom Innovations Inc. (Switzerland).

References

[1] E. D. de Jong. The incremental pareto-coevolution archive. In *GECCO (1)*, volume 3102 of *Lecture Notes in Computer Science*, pages 525–536. Springer, 2004.

[2] G. Folino, C. Pizzuti, and G. Spezzano. Ensemble techniques for parallel genetic programming based classifiers. In *Proceedings of the 6th European Conference on Genetic Programming (EuroGP)*, volume 2610 of *Lecture Notes in Computer Science*, pages 59–69. Springer, 2003.

[3] C. Gathercole and P. Ross. Dynamic training subset selection for supervised learning in genetic programming. In *PPSN*, volume 866 of *Lecture Notes in Computer Science*, pages 312–321. Springer, 1994.

[4] E. D. Jong. Towards a bounded pareto-coevolution archive. In *Proceedings of the Congress on Evolutionary Computation CEC-04*, volume 2, pages 2341–2348, 2004.

[5] H. Juille and J. Pollack. Massively parallel genetic programming. In *Advances in Genetic Programming, 2nd ed.*, pages 339–358. MIT Press, Cambridge, MA, USA, 1996.

[6] J. Knowles and D. Corne. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimization. In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 98–105, Mayflower Hotel, Washington D.C., USA, 6-9 1999. IEEE Press.

[7] J. R. Koza. *Genetic Programming II.* Cambridge, Mass.:MIT Press, 1994.

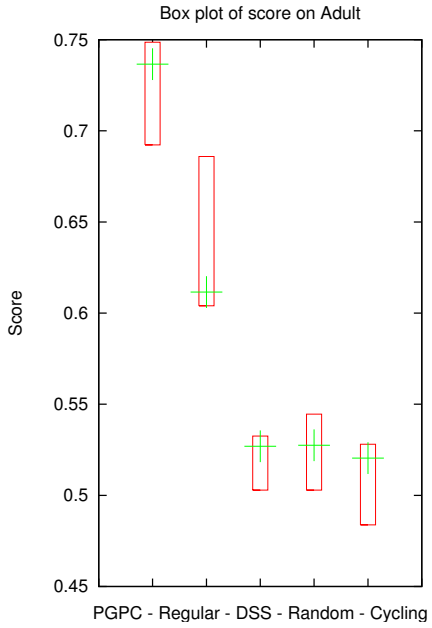
[8] C. Lasarczyk, P. Dittrich, and W. Banzhaf. Dynamic subset selection based on a fitness case topology. *Evolutionary Computation*, 12(2):223–242, 2004.

[9] P. Nordin. A compiling genetic programming system that directly manipulates the machine code. In *Advances in Genetic Programming.*, pages 311–334. MIT Press, Cambridge, MA, USA, 1994.

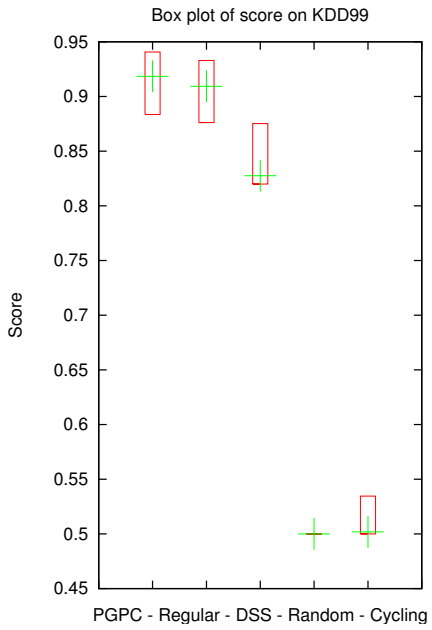
[10] P. Nordin and W. Banzhaf. An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behavior.*, 5(2):107–140, 1996.

[11] D. Song, M. Heywood, and A. Zincir-Heywood. Training genetic programming on half a million patterns: An example from anomaly detection. *IEEE Transactions on Evolutionary Computation*, 9(3):225–239, 2005.

[12] E. Zitzler and L. Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.



(a) Adult



(b) KDD99

Figure 2: Test data box plot as evaluated using the 'score' metric. Note: the variance of the Random algorithm on KDD99 is minimal, with the difference between the minimum and maximum value being 0.005215 percent, yielding an insignificant box plot.