Linear Genetic Programming for Multi-class Object Classification

Christopher Fogelberg and Mengjie Zhang

School of Mathematics, Statistics and Computer Sciences
Victoria University of Wellington, P. O. Box 600, Wellington, New Zealand
{fogelbchri,mengjie}@mcs.vuw.ac.nz

Abstract. Multi-class object classification is an important field of research in computer vision. In this paper basic linear genetic programming is modified to be more suitable for multi-class classification and its performance is then compared to tree-based genetic programming. The directed acyclic graph nature of linear genetic programming is exploited. The existing fitness function is modified to more accurately approximate the true feature space. The results show that the new linear genetic programming approach outperforms the basic tree-based genetic programming approach on all the tasks investigated here and that the new fitness function leads to better and more consistent results. The genetic programs evolved by the new linear genetic programming system are also more comprehensible than those evolved by the tree-based system.

1 Introduction

Image classification tasks occur in a wide variety of problem domains. While human experts can frequently accurately classify the data manually, such experts are typically rare or too expensive. Thus computer based solutions to many of these problems are very desirable.

Genetic Programming (GP) [1, 2] is a promising approach for building reliable classification programs quickly and automatically, given only a set of examples on which a program can be evaluated. GP uses ideas analogous to biological evolution to search the space of possible programs to evolve a good program for a particular task.

While showing promise, current GP techniques frequently do not give satisfactory results on difficult classification tasks, particularly multi-class classification (tasks with more than two classes). There are at least two limitations in currently used GP *program structures* and *fitness functions* that prevent GP from finding acceptable programs in a reasonable time.

The programs that GP evolves are typically tree-like structures [3], which map a vector of input values to a single real-valued output[4–6]. For classification tasks, this output must be mapped into a set of class labels. For binary classification problems, there is a natural mapping of negative values to one class and positive values to the other class. For multi-class classification problems, finding the appropriate boundaries on the number line to separate the classes is very difficult. Several new translations have recently been developed in

the interpretation of the single output value of the tree-based GP [4, 7, 8], with differing strengths in addressing different types of problem. While these translations have achieved better classification performance, the evolution is still slow and the evolved programs are hard to interpret, particularly for more difficult problems or problems with a large number of classes.

In solving classification problems, GP typically uses the classification accuracy, error rate or a similar measure as the fitness function [5, 7, 8], which approximates the true fitness of an individual program. Given that the training set size is often highly limited, such an approximation frequently fails to accurately estimate the classification of the true feature space.

1.1 Goals

To address the problems above, this paper aims to investigate an approach to the use of linear genetic programming (LGP) and a new fitness function for multi-class object classification problems. This approach will be compared with the basic tree-based GP (TGP) approach on three image classification tasks of increasing difficulty. Specifically, we are interested in:

- Whether the LGP approach outperforms the basic TGP approach on these object classification problems in terms of classification performance.
- Whether the genetic programs evolved by LGP are more comprehensible.
- Whether the new fitness function improves the classification performance over the existing fitness function.

2 LGP for Multi-class Object Classification

2.1 LGP Overview

This work used register machine LGP (hereafter just LGP) [2], where an individual program is represented by a sequence of register machine instructions, typically expressed in human-readable form as C-style code.

Prior to any program being executed, the registers which it can read from or write to are zeroed. The features representing the objects to be classified are loaded into predefined registers. The program is executed in an imperative manner and represents a *directed acyclic graph* (DAG). This is different from tree-based GP which represents a tree. Any register's value may be used in multiple instructions during the execution of the program.

2.2 Multi-class Output Interpretation

An LGP program often has only one register interpreted in determining its output [9, 2]. This configuration can be easily used for regression and binary classification problems as in the tree-based GP.

In this work, we use LGP for multi-class object classification problems. We want an LGP program to produce one output for each class. Thus, instead of using only one register as the output, we use multiple registers each corresponding

to one class. The winner-takes-all strategy is then used and the class represented by the register with the largest value is considered the class of the input object by that genetic program.

This program output representation for the different classes is very similar to a feed forward neural network classifier [10]. However, the structure of such an LGP program is more flexible than that of the feed forward neural network.

2.3 Evolutionary Operators

We used reproduction, crossover and mutation as genetic operators. In reproduction, the best programs in the current generation are copied into the next generation without any change.

We used two different forms of mutation [11] in this work. *Macromutation* involves the replacement of an entire instruction with a randomly generated one. *Micromutation* changes only either the destination register, a source register or the operation. These operations can cause dramatic changes in the DAG that a program represents [12].

In the crossover operator, we randomly choose a section from each of the two parents, then swap them to produce offspring. If a newly produced program is longer than the maximum length allowed, then an instruction is randomly selected and removed until the program can fit into the maximum length. This is similar to two-point crossover in GAs[13], but the two sections chosen from the parents can have different lengths here.

2.4 The Old Fitness Function and the Hurdle Problem

Given that the size of the training set must be finite, any fitness function can only be an approximation to an program's true fitness. This can lead to problems such as overfitting, where a program's true fitness is sacrificed for fitness on the training set. In a multi-class object classification problem, a program's true fitness is the fraction of the feature space it can correctly classify. A good fitness function is one which accurately estimates this fraction.

A typical fitness function for classification problem is the *error rate* of a program classifier. This was also used in our early experiments. While it performed reasonably well, this fitness function frequently fails to accurately estimate the fraction of the feature space correctly classified by a program.

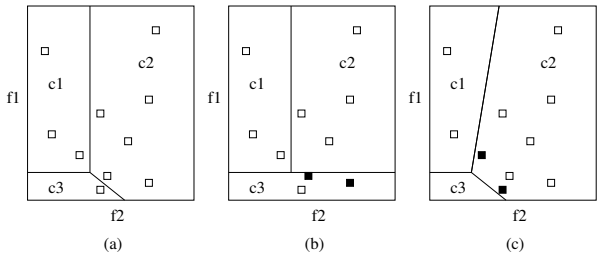


Fig. 1. The hurdle problem. Solid objects are misclassified.

Figure 1 shows a simple classification problem with two features. Figure 1(a) shows the true feature space — feature vectors of class c1 objects always appear

in the fraction of the feature space denoted "c1", and similarly for the fractions denoted "c2" and "c3". Figure 1(b) shows that `program1` misclassifies two objects of c2 as c3. This program has an error rate of 18% (2/11). Figure 1(c) shows that `program2` misclassifies one object from class c3 and one object from class c1 as c2. This program also has an error rate of 18% and will be treated the same as the `program1`. `program2` actually approximated the true fitness more accurately than `program1`, but the fitness function cannot accurately reflect this difference.

We call this problem *the hurdle problem*. It occurs when (any two) classes have a very complex boundary in the feature space. In such a situation, it is easy to classify the bulk of fitness cases for one class correctly, but learning to recognise the other class often initially comes only at an equal or greater loss of accuracy in classifying the first class. This creates a strong selection pressure against making the classification boundary in the feature space more complex and GP with such a fitness function often cannot surmount the hurdle.

2.5 The Decay Curve Fitness Function

To address the hurdle problem, we introduced a new fitness function, the *decay curve fitness function* to estimate true fitness more accurately. The new fitness function uses an *increasing penalty* for each of the M_c misclassifications of some class c, as shown in equation 1.

$$f_{decay} = \sum_{c} \sum_{i=0}^{M_c} \alpha^{\beta i} / N \tag{1}$$

The values of α and β are determined through empirical search. We used a fitness function with $\alpha > 1$ to approximate the true fitness so that the penalty of later misclassifications increased exponentially. N is the number of training examples.

Obviously, as α approaches 1.0 and β approaches 0.0, the curve becomes progressively flatter and more similar to a traditional fitness function (error rate in this case).

3 Experiment Design and Configuration

3.1 Data Sets

Experiments were conducted on three different image data sets providing object classification problems of increasing difficulty. Examples are shown in figure 2.

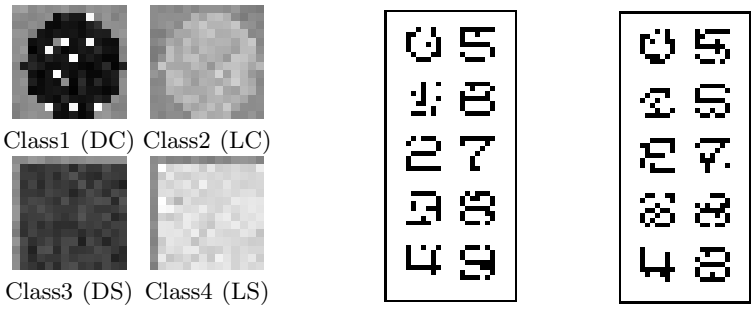


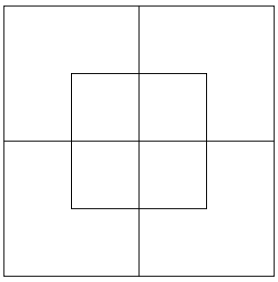
Fig. 2. Image data sets. (a) shape; (b) digit15; (c) digit30.

The first data set (figure 2a) was generated to give well defined objects against a relatively clean background. The pixels of the objects were produced using a Gaussian generator with different means and variances for each class. Four classes of 600 small objects (150 for each class) were used to form the classification data set. The four classes are: dark circles (*class1*), light circles (*class2*), dark squares (*class3*), and light squares (*class4*). This data set is referred to as *shape*. The objects in class1 and class3, and in class2 and class4 are very similar in the average values of pixel intensities, which makes the problem reasonably difficult.

The second and third data sets are two digit recognition tasks, each consisting of 1000 digit examples. Each digit is represented by a 7×7 bitmap image. In the two tasks, the goal is to automatically recognise which of the 10 classes (0, 1, 2, ..., 9) each bitmap belongs to. Note that all the digit patterns have been corrupted by noise. In the two tasks (figure 2 (b) and (c)), 15% and 30% of pixels, chosen at random, have been flipped. In data set 2 (*digit15*), while some patterns can be clearly recognised by human eyes such as "0", "2", "5", "7", and possibly "4", it is not easy to distinguish between "6", "8"and "3". The task in data set 3 (*digit30*) is even more difficult — human eyes cannot recognise majority of the patterns, particularly "8", "9" and "3", "5" and "6", and even "1", "2" and "0". In addition, the number of classes is much greater than that in task 1, making the two tasks even more difficult.

3.2 Primitive Sets

Terminals. In the *shape* data set, we used eight features extracted from the objects and an random number as the terminal set. The eight features are shown in figure 3.



Feature	LGP Index	Description
f1	`cf[0]`	mean brightness of the entire object
f2	`cf[1]`	mean of the top left quadrant
f3	`cf[2]`	mean of the top right quadrant
f4	`cf[3]`	mean of the bottom left quadrant
f5	`cf[4]`	mean of the bottom right quadrant
f6	`cf[5]`	mean of the centre quadrant
f7	`cf[6]`	standard deviation of the whole object
f8	`cf[7]`	standard deviation of centre quadrant

Fig. 3. Terminal set for the *shape* data set.

For the two digit data sets, we used the raw pixels as the terminal sets, meaning that the feature vector of each object has 49 values. The large number of terminals makes these tasks more difficult, but we expect that the GP evolutionary process can automatically select those highly relevant to each recognition problem.

Functions. The function set for all the three data sets was {`+, -, *, /, if`}. Division (/) was protected to return 0 on a divide-by-zero. `if` executes the next statement if the condition is true.

3.3 Parameters and Termination Criteria

The parameter values used for the LGP system for the three data sets are shown in table 1. Evolution is terminated at generation 50 unless a successful solution is found, in which case the evolution is terminated early.

Table 1. Parameter values for the LGP system for the three data sets.

parameter name	shape	digit15	digit30	parameter name	shape	digit15	digit30
pop_size	500	500	500	macromutation_rate	30%	30%	30%
max_program_length	15	35	35	micromutation_rate	30%	30%	30%
reproduction_rate	10%	10%	10%	α	1.2	1.2	1.2
crossover_rate	30%	30%	30%	β	0.24	0.24	0.24

3.4 TGP Configuration

The LGP approach developed in this work was compared to the basic TGP approach [3]. In TGP, the ramped half-and-half method was used for initial generation and mutation[2]. The proportional selection mechanism and the reproduction, crossover and mutation operators [3] were used in the learning and evolutionary process. The program output was translated into a class label according to the static range selection method [4].

The TGP system used the same terminal sets, function sets, fitness function, population size and termination criteria for the three data sets as the LGP approach. The reproduction, mutation, and crossover rates used were 10%, 30%, and 60%, respectively. The program depth was 3–5 for the shape data set, and 4–6 for the two digit data sets. All single experiments were repeated 50 times. The average results are presented in the next section.

The program depths above in TGP were derived from the LGP program lengths based on a heuristic. An LGP instruction typically consists of one or two arguments and an operation, each corresponding to a node in a TGP program tree. Considering that each TGP operation might be used by its children and/or parents, an LGP instruction roughly corresponds to 1.5 tree nodes. Assuming each non-leaf node has two children (or more for some functions), we can calculate the expressive capacity of a depth-n TGP in LGP program instructions.

4 Results and Discussion

4.1 Classification Performance

Classification Accuracy. Table 2 shows a comparison between the LGP approach developed in this work and the standard TGP approach for the three object classification problems.

On the shape data set, our LGP approach always generated a genetic program which successfully classified all objects in the training set. These 50 program classifiers also achieved almost perfect classification performance on the unseen objects in the test set. On the other hand, the TGP approach only achieved about 85.04% and 84.41% accuracy on the training and the test sets, respectively. In addition, the LGP approach resulted in a much smaller standard deviation than

Table 2. Classification accuracy of the LGP and TGP on the three data sets.

Data set	Method	Training Set Accuracy % ($\mu \pm \sigma$)	Test Set Accuracy % ($\mu \pm \sigma$)
shape	LGP	100.00 ± 0.00	99.91 ± 0.17
	TGP	85.04 ± 16.49	84.41 ± 17.17
digit15	LGP	68.02% ± 4.16%	62.48% ± 5.03%
	TGP	52.60% ± 6.65%	51.80% ± 6.85%
digit30	LGP	55.22% ± 3.49%	51.04% ± 4.26%
	TGP	41.15% ± 5.03%	35.00% ± 6.17%

the TGP approach. This shows that the LGP method is more stable and more reliable than the TGP approach on this problem. These results suggest that the LGP approach greatly outperforms the TGP approach on this data set in terms of the classification accuracy.

The classification results on the two digit data sets show a similar pattern to those on the shape data set. In both cases, the LGP approach achieved a higher average value and a lower standard deviation of the classification accuracy on the test set than the corresponding TGP approach. The improvements are quite considerable, suggesting that the LGP approach is better than the TGP approach for these multi-class object classification problems.

Training Efficiency. Inspection of the number of generations used reveals that the LGP approach is more efficient than the TGP approach in finding a good genetic program classifier for these object classification problems. For example, in the shape data set, the $\mu \pm \sigma$ of the number of generations for the LGP approach was 16.46 ± 10.22, which was much smaller than the corresponding number for the TGP approach (41.22 ± 14.11).

4.2 Comprehensibility of the Evolved Genetic Programs

To check whether the genetic programs evolved by the LGP approach are easy to interpret or not, we use a typical evolved program which perfectly classified all objects for the shape data set as an example. The code of the evolved genetic program is shown in figure 4 (left). Note that structural introns are commented using `//`. The DAG representation of the simplified program is shown in figure 4 (right) after the introns are removed.

```
//r[1] = r[1] / r[1];
//r[3] = cf[0] + cf[5];
//if(r[3] < 0.86539)
//r[3] = r[3] - r[1];
r[0] = 0.453012 - cf[1];
//r[3] = r[2] * cf[5];
r[1] = r[0] * 0.89811;
if(cf[6] < cf[1])
r[2] = 0.453012 - cf[3];
r[3] = cf[4] - 0.86539;
```

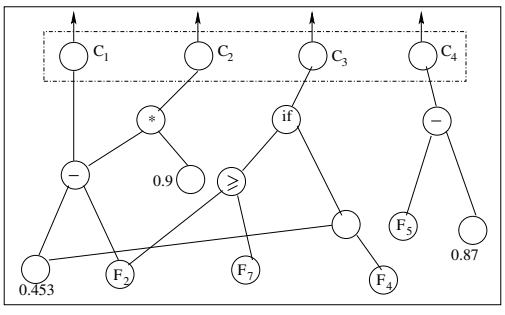


Fig. 4. A sample program evolved by LGP.

In this program, the array `cf (cf[0] to cf[7])` are the eight feature terminals (`f1, ..., f8`) as described in figure 3 and the register array `r (r[0] to r[3])` correspond to the four class labels (`class1, class2, class3, class4`). Given an object, the feature values and the register values can be easily calculated and the class of the object can be simply determined by taking the register with the largest value. For example, given the following four objects with different feature values:

```
                cf[0]   cf[1]   cf[2]   cf[3]   cf[4]   cf[5]   cf[6]   cf[7]
-------------------------------------------------------------------------------
Obj1 (class1): 0.3056 0.3458 0.2917 0.2796 0.3052 0.1754 0.5432 0.5422
Obj2 (class2): 0.6449 0.6239 0.6452 0.6423 0.6682 0.7075 0.1716 0.1009
Obj3 (class3): 0.2783 0.3194 0.2784 0.2770 0.2383 0.2331 0.2349 0.0958
Obj4 (class4): 0.8238 0.7910 0.8176 0.8198 0.8666 0.8689 0.2410 0.1021
```

we can obtain the following register values and classification `r[]` for each object example.

Object	True-Class	r[0]	r[1]	r[2]	r[3]	Classified-Class
Obj1	class1	**0.1474**	0.13240	0.0000	-0.5602	class1
Obj2	class2	-0.1919	**-0.1723**	-0.1893	-0.1972	class2
Obj3	class3	0.1747	0.1569	**0.1760**	-0.6271	class3
Obj4	class4	-0.3708	-0.3330	-0.3668	**0.0012**	class4

As can be seen from the results, this genetic program classified all the four object examples correctly. Examining the program and features used suggests that the genetic programs evolved by LGP are quite comprehensible.

Further inspection of this program reveals that only four of eight features were selected from the terminal set. This suggests that the LGP approach can automatically select features relevant to a particular task. The DAG representation of the program (figure 4b) shows that the LGP approach can co-evolve sub-programs together each for a particular class and that some terminals and functions can be reused by different sub-programs.

On the other hand, a program evolved by the TGP approach can only produce a single value, which must be translated/interpreted into a set of class labels. A typical genetic program evolved by the TGP approach is:

```
(* (- (+ (/ f1 -0.268213) (/ -0.828695 f6))
      (/ (/ f7 f6) (+ -0.828695 f5)))
   (* (- (/ f1 f5) (/ f5 f6))
      (+ (- f4 -0.828695) (+ f1 f2)))
)
```

This program used almost all the features and it is not clear how it does the classification. Such programs are more difficult to interpret for multi-class classification problems.

4.3 Impact of the New Fitness Function

To investigate whether the new fitness function is helpful in reducing the hurdle problem, we used the shape data set as an example to compare the classification

performance between the new fitness function and the old fitness function (error rate).

When doing experiments, we used a slightly different setting in program size. Notice that the frequency of the hurdle problem will drop as the program size is increased, although it is not eliminated. Hence the LGP programs in the assessment of the new decay curve fitness function use a program length 10, which is still long enough to express a solution to the problem — solutions have been found when the maximum length is 5. In TGP the tree depths are left at 3–5. These limits are likely to be representative of the situation when a much more difficult problem is being addressed. In such tasks, the maximum depth which is computationally tractable with existing hardware may also be so short relative to the problem's difficulty that the hurdle problem is a major issue.

Table 3 shows the classification results of the two fitness functions using both the TGP and the LGP methods for the shape data set. For the TGP method, the new fitness function led to a very significant improvement on both the training set and the test set. For the LGP method, the classification accuracy was also improved using the new fitness function, but the improvement was not as significant. This was mainly because the LGP method with the old fitness function already performed quite well (98.76%) due to the power of LGP. When using either the old or the new fitness functions, the LGP method always outperformed the TGP method. This is consistent with our previous observation.

Table 3. A comparison of the two fitness functions on the shape data set.

Method	Fitness Function	Training Accuracy ($\mu \pm \sigma$)	Test Accuracy ($\mu \pm \sigma$)
TGP	old	77.31% ± 6.74%	77.14% ± 6.68%
	new	85.04% ± 16.49%	84.41% ± 17.17%
LGP	old	98.90% ± 4.98%	98.76% ± 5.04%
	new	99.97% ± 0.11%	99.90% ± 0.25%

Further inspection of the results using the TGP method on the shape data set shows that only 6 of the 50 runs using the old fitness function had a test or training accuracy greater than 75%. When those 6 runs are excluded, the μ and σ becomes 74.95% ± 0.0019% on the training set and 74.86% ± 0.0024% on the test set. These figures indicate how solid the hurdle actually is in situations where the problem is at the limit of a GP configuration's expressiveness. By using the new decay curve fitness function, 36 of the 50 runs finished with test and training accuracies greater than 75%.

5 Conclusions

The goal of this paper was to investigate an approach to the use of LGP and a new fitness function for multi-class object classification problems. This approach was compared with the basic TGP approach on three image data sets providing object classification problems of increasing difficulty. The results suggest that the LGP approach outperformed the TGP approach on all tasks in terms of classification accuracy and evolvability.

Inspection of the evolved genetic programs reveals that the programs evolved by the LGP approach are relatively easy to interpret for these problems. The results suggest that the LGP approach can automatically select features relevant to a particular task, that the programs evolved by LGP can be represented as a DAG, and that the LGP approach can simultaneously sub-programs together, each for a particular class.

A comparison between the old fitness function and the new fitness function has also highlighted the nature of the fitness function as an approximation to the true fitness of a problem. The results show that the new fitness function, with either the TGP approach or the LGP approach, can bring better and more consistently accurate results than the old fitness function.

Although developed for multi-class object classification problems, we expect that this approach can be applied to other multi-class classification problems.

References

1. Koza, J.R.: Genetic Programming. MIT Press, Campridge, Massachusetts (1992)
2. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications. Morgan Kaufmann, dpunkt.verlag (1998)
3. Koza, J.R.: Genetic Programming II: Automatic Discovery of Reusable P rograms. Cambridge, Mass. : MIT Press, London, England (1994)
4. Loveard, T., Ciesielski, V.: Representing classification problems in genetic programming. In: Proceedings of the Congress on Evolutionary Computation. Volume 2., IEEE Press (2001) 1070–1077
5. Tackett, W.A.: Recombination, Selection, and the Genetic Construction of Computer Programs. PhD thesis, Faculty of the Graduate School, University of Southern C alifornia, Canoga Park, California, USA (1994)
6. Zhang, M., Ciesielski, V.: Genetic programming for multiple class object detection. In Proceedings of the 12th Australian Joint Conference o n Artificial Intelligence, Springer-Verlag (1999) 180–192 (LNAI Volume 1747).
7. Zhang, M., Ciesielski, V., Andreae, P.: A domain independent window-approach to multiclass object detection using genetic programming. EURASIP Journal on Signal Processing **2003** (2003) 841–859
8. Zhang, M., Smart, W.: Multiclass object classification using genetic programming. In Applications of Evolutionary Computing, EvoWorkshops2004. Volume 3005 of LNCS., Springer Verlag (2004) 369–378
9. Oltean, M., Grosan, C., Oltean, M.: Encoding multiple solutions in a linear genetic programming chromosome. In Proceedings of 4th International Conference on Computational Science, Part III. Springer-Verlag (2004) 1281–1288
10. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. In Parallel distributed Processing, Explorations in the Microstructure of Cognition, Volume 1: Foundations. The MIT Press (1986)
11. Brameier, M., Banzhaf, W.: A comparison of genetic programming and neural networks in medical data analysis. Reihe CI 43/98, Dortmund University (1998)
12. Brameier, M., Banzhaf, W.: Effective linear genetic programming. Technical report, Department of Computer Science, University of Dortmund, Germany (2001)
13. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison–Wesley, Reading, MA (1989)