Connection Science
Vol. 19, No. 4, December 2007, 339–359

Taylor & Francis
Taylor & Francis Group

A linear structured approach and a refined fitness function in genetic programming for multi-class object classification

MENGJIE ZHANG*†‡, CHRISTOPHER GRAEME FOGELBERG† and YUEJIN MA‡

†School of Mathematics, Statistics and Computer Sciences, Victoria University of Wellington, PO Box 600, Wellington, New Zealand
‡College of Mechanical and Electrical Engineering, Agricultural University of Hebei, 071001 Baodong, China

This paper describes an approach to the use of genetic programming (GP) to multi-class object recognition problems. Rather than using the standard tree structures to represent evolved classifier programs which only produce a single output value that must be further translated into a set of class labels, this approach uses a linear structure to represent evolved programs, which use multiple target registers each for a single class. The simple error rate fitness function is refined and a new fitness function is introduced to approximate the true feature space of an object recognition problem. This approach is examined and compared with the tree based GP on three data sets providing object recognition problems of increasing difficulty. The results show that this approach outperforms the standard tree based GP approach on all the tasks investigated here and that the programs evolved by this approach are easier to interpret. The investigation into the extra target registers and program length results in heuristic guidelines for initially setting system parameters.

Keywords: Linear genetic programming; Program structure; Program representation; Fitness function; Multi-class classification; Object classification; Object recognition

1. Introduction

Object recognition tasks occur in a wide variety of problem domains. Detecting faces from video images, finding tumours in a database of x-ray images, and recognising digits in the postal/zip code images are just three examples. In many cases, people (possibly highly trained experts) are able to perform the object recognition task well, but there is either a shortage of such experts, or the cost of people is too high. Given the amount of image data containing objects of interest that need to be recognised and classified, computer based solutions to many of these problems are very desirable.

An object recognition/classification program must automatically and correctly map an input vector describing an object image instance to a small set of class labels. Writing such programs is usually difficult and often infeasible: human programmers often cannot identify all the subtle conditions needed to distinguish between all object instances of different classes.

*Corresponding author. Email: mengjie@mcs.vuw.ac.nz

Connection Science
ISSN 0954-0091 print/ISSN 1360-0494 online © 2007 Taylor & Francis
http://www.tandf.co.uk/journals
DOI: 10.1080/09540090701725557

Genetic Programming (GP) (Koza 1992; Banzhaf *et al.* 1998) is a promising approach for building reliable classification programs quickly and automatically, given only a set of example data 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. A strength of this approach is that evolved programs can be much more flexible than the highly constrained, parameterised models used in other techniques such as neural networks and support vector machines. GP has been applied to a range of object classification tasks with some success (Gray 1997; Howard *et al.* 2002; Olague *et al.* 2006; Krawiec and Bhanu 2005).

While showing promise, current GP techniques frequently do not give satisfactory results on difficult object recognition tasks, particularly those with multiple classes (tasks with three or more classes). There are at least two limitations in current GP *program structures* and *fitness functions* used in these classification systems that prevent GP from finding acceptable programs in a reasonable time.

The programs that GP evolves are typically tree-like structures (Koza 1994), which map a vector of input values to a single real-valued output (Loveard and Ciesielski 2001a; Tackett 1994). For object recognition/classification tasks, this output must be translated 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 problems, finding the appropriate boundaries on the numeric value to separate the different classes well is very difficult. Several new translation rules have recently been developed in the interpretation of the single output value of the tree based GP (Loveard and Ciesielski 2001a; Zhand and Smart 2004), with differing strengths in addressing different types of problem. While these translations have achieved better classification performance, the evolution is generally slow and the evolved programs are still hard to interpret, particularly for more difficult problems or problems with a large number of classes.

In dealing with object recognition/classification problems, GP typically uses classification accuracy, error rate or a similar measure as the fitness function (Tackett 1994; Zhang *et al.* 2003), which approximates the true fitness of an individual program. Given that the training set size is often limited, such an approximation frequently fails to accurately estimate the classification of the true feature space.

1.1 *Goals*

To avoid these problems, this paper aims to investigate an approach to the use of linear genetic programming (LGP) structure and a refined fitness function for multi-class object recognition problems. This approach will be examined and compared with the basic tree based GP (TGP) approach on three image classification tasks of increasing difficulty. Specifically, we are interested in investigating the following research issues.

- How the LGP approach deals with multi-class classification problem.
- 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 easier to interpret or more natural to multi-class object classification than the TGP programs.
- Whether the new fitness function improves the object classification performance over the existing fitness function.
- Whether the use of extra target registers in the linear genetic programs improves the system performance.
- How the program length affects the object classification performance.

1.2 *Organisation*

The rest of the paper is organised as follows. Section 2 gives essential background of object recognition and GP related work. Section 3 describes the LGP approach to multi-class object classification/recognition. A problem faced by the existing fitness functions is described and a new fitness function which addresses this problem is proposed. The experiment design and configurations are described in section 4. Section 5 presents a series of results, comparisons, and analyses on the new linear structured GP and TGP, the proposed fitness function, evolved genetic programs, program length and extra target registers for multi-class object classification. The conclusions are summarised in section 6 with future work directions.

2. Background

2.1 *Object recognition*

Object Recognition, also known as automatic object recognition or automatic target recognition, is a specific field and a challenging problem in computer vision and image understanding (Forsyth and Ponce 2003). This task often involves *object localisation* and *object classification*. Object localisation refers to the task of identifying the positions of the objects of interest in a sequence of images either within the visual or infrared spectral bands. Object classification refers to the task of discriminating between images of different kinds of objects, where each image contains only one of the objects of interest. Object localisation can also be considered a kind of object classification task – binary classification, where the two classes are *object* and *background*, but the examples of the two classes are highly unevenly distributed.

Traditionally, most research on object recognition involves four stages: *preprocessing, segmentation, feature extraction* and *classification* (Caelli and Bischof 1997). The preprocessing stage aims to remove noise or enhance edges. In the segmentation stage, a number of coherent regions and 'suspicious' regions which might contain objects are usually located and separated from the entire images. The feature extraction stage extracts domain specific features from the segmented regions. Finally, the classification stage uses these features to distinguish the classes of the objects of interest. The features extracted from the images and objects are generally domain specific such as high level relational image features. Data mining and machine learning algorithms are usually applied to object classification.

Object recognition has been of tremendous importance in many application domains. These domains include military applications (Wong and Sundareshan 1998), human face recognition (Teller and Veloso 1995a), agricultural product classification (Winter *et al.* 1996), handwritten character recognition (Lecun *et al.* 2001), medical image analysis (Verma 1998), postal code recognition (de Ridder *et al.* 1996), and texture classification (Song 2003).

Since the 1990s, many methods have been employed for object recognition. These include different kinds of neural networks (Azimi-Sadjadi *et al.* 2000), genetic algorithms (Bala *et al.* 1997), decision trees (Russell and Norvig 2003), statistical methods such as Gaussian models and Naïve Bayes (Dunham 2003), support vector machines (Dunham 2003), GP (Howard *et al.* 1999), and hybrid methods (Yao and Liu 1997).

2.2 *GP related work to object recognition*

Since the early 1990s, there has been only a relatively small amount of work on applying GP techniques to object recognition, including object classification and object localisation.

Tackett (1993, 1994) uses GP to assign detected image features to a *target* or *non-target* category. Seven primitive image features and 20 statistical features are extracted and used as the terminal set. The four standard arithmetic operators and a logic function are used as the function set. The fitness function is based on the classification result. The approach was tested on US Army NVEOD Terrain Board imagery, where vehicles such as tanks need to be classified. The GP method outperformed both a neural network classifier and a binary tree classifier on the same data, producing lower rates of false positives for the same detection rates.

Koza (1994) (in chapter 15) uses a 'turtle' to walk over a bitmap landscape. This bitmap is to be classified either as a letter 'L', a letter 'I', or neither of them. The turtle has access to the values of the pixels in the bitmap by moving over them and calling a detector primitive. The turtle uses a decision tree process, in conjunction with negative primitives, to walk over the bitmap and decide which category a particular landscape falls into. Using automatically defined functions as local detectors and a constrained syntactic structure, some perfect scoring classification programs were found. Further experiments showed that detectors can be made for different sizes and positions of letters, although each detector has to be specialised to a given combination of these factors.

Loveard and Ciesielski (2001a) use strongly typed GP for a number of object recognition problems, including classification of medical images and satellite image objects. This work investigates a number of classification strategies. Their results show that the dynamic range selection strategy outperformed other strategies on those classification problems.

Andre (1994) uses GP to evolve functions that traverse an image, calling upon co-evolved detectors in the form of hit-miss matrices to guide the search. These hit-miss matrices are evolved with a two-dimensional genetic algorithm. These evolved functions are used to discriminate between two letters or to recognise single digits.

Song *et al.* (Song 2003; Song and Ciesielski 2004) use tree based GP for a series of object image texture classification problems, such as classification of bitmap patterns, Brodatz textures and mashing images. This work mainly focuses on the use of GP for binary classification problems. There are also some multi-class problems in this work but he decomposes the multi-class problems into multiple binary classification problems then applies GP to these binary problems. Both features and pixel values are used as terminals, the four standard arithmetic operators and a conditional operator with some relational operators are used to construct the function set, and the classification accuracy is used as the fitness function. The results show that, 'with an appropriate methodology, GP can be used as a texture classification method without computationally expensive feature extraction'.

Robinson and McIlroy (1995) apply GP techniques to the problem of eye location in grey-level face images. The input data from the images is restricted to a 3000 pixel block around the location of the eyes in the face image. This approach produced promising results over a very small training set, up to 100% true positive detection with no false positives, on a three image training set. Over larger sets the GP approach performed less well however, and could not match the performance of neural network techniques.

Ciesielski *et al.* (2005) use GP for a real world object detection problem – finding orthodontic landmarks in cranio-facial X-Rays. The system could evolve genetic programs to implement a linear function of the features. Analysis of these linear functions reveals that underlying regularities can be captured. The analysis also suggests that evolved algorithms are a realistic solution to the object detection problem, given the features and operators available.

Winkeler and Manjunath (1997) produce genetic programs to locate faces in images. Face samples are cut out and scaled, then pre-processed for feature extraction. The statistics gleaned from these segments are used as terminals in GP which evolves an expression returning how likely a pixel is to be part of a face image. Separate experiments process the grey scale image directly, using low level image processing primitives and scale-space filters.

Teller and Veloso (1995a) use a GP method based on the PADO language to perform face recognition tasks on a database of face images in which the evolved programs have a local indexed memory. The approach was tested on a discrimination task between 5 classes of images (Teller and Veloso 1995b) and achieved up to 60% correct classification for images without noise.

Zhang *et al.* (Smart and Zhang 2003; Zhang and Smart 2006) use GP for a number of object classification and detection problems. Typically, low level pixel statistics are used to form the terminal set, the four arithmetic operators are used to construct the function set, and the fitness functions are based on either classification accuracy or error rate for object classification problems, and detection rate and false alarm rate for object localisation and detection problems. Good results have been achieved on classification and detection of regular objects against a relatively uncluttered background.

Bhanu *et al.* (2005) investigate evolutionary computational techniques including tree-based GP, LGP, coevolutionary GP and genetic algorithms to automate the synthesis and analysis of object detection and recognition systems. This book also includes evolutionary feature synthesis and selection techniques for object detection and recognition.

More detailed work about GP and other evolutionary computation techniques for object recognition, image analysis and computer vision applications can be seen from a recent journal special issue (Olague *et al.* 2006) and a recent book (Cagnoni *et al.*2007).

Among these GP works done for object recognition and classification, most of them used the tree based GP and a few used linear structured GP and grammar based GP. This paper investigates a linearly structured GP approach and compares this approach with the tree-based GP approach.

2.3 *GP for multi-class classification and fitness functions*

The GP related work to multi-class classification can be categorised into three major approaches. The first approach is to convert the single multi-class problem into multiple binary classification sub-problems and each single binary classification subproblem is solved using a single GP run (Kishore *et al.* 2000; Loveard and Ciesielski 2001a). This includes the `one` *vs* `many` method where the number of GP runs is the number of classes, and the `one` *vs* `one` method, where the number of GP runs is the combination of every two classes out of the total number of classes. In both cases, the fitness function in GP is simply focused on a single binary classification task, which is known as an easy task for both the basic TGP and LGP. The major disadvantage is that this approach needs this multi-to-binary conversion, needs multiple GP evolutions, and needs to consider and integrate all the evolved programs together for classifying unseen examples.

The second approach is similar to the first approach, except that all the programs are co-evolved in a single GP run (Muni *et al.* 2004; Smart and Zhang 2005). In this case, the fitness function is generally more complex than the first approach since the single fitness function has to deal with all the binary classification subproblems. As this approach still needs multiple programs, the fitness function needs to explicitly combine them together to solve the entire multi-class classification program.

The third approach is to use a single genetic program for the entire multi-class problem (Loveard Ciesielski 2001a; Zhang and Smart 2006). In this case, the single program will need to have directly corresponding relationships with all the classes, and the fitness function needs to include the heuristics that can help evolve programs that map the fitness cases in different classes into the correct class labels. A major advantage of this approach is that only a single GP program is needed and accordingly the efficiency is often better compared with the first

approach. Another advantage is that the conversion from multi-class to binary class is removed. A major disadvantage of this approach is that a single output either from the root node in the basic TGP or from the single target register in the basic LGP must be translated into a set of class labels, as discussed in the introduction.

This work is focused on the third approach targeting the use of a single GP program for the entire multi-class object classification problems. Some improvements to the fitness function have been made, including the dynamic range selection (Loveard and Ciesielski, 2001a), unbalanced-classes based evaluation (Howard *et al.* 2002), the probability based methods (Zhang and Smart, 2006), and weighted frequency based evaluation (Paul and Iba 2006). This work considers another new fitness function, which will be described in section 3.3.

3. Linear structured GP for multi-class object recognition/classification

3.1 *Program representation and structure*

Based on the ideas of the register machine LGP (Banzhaf *et al.* 1998), a new LGP system called `VUWLGP` has been recently developed (Fogelberg and Zhang 2005). In the VUWLGP system, an individual program is represented by a sequence of register machine instructions, typically expressed in human-readable form as C-style code. Each instruction typically has three components: *source registers* corresponding to features of a particular task or some random constant values generated during the evolutionary process, a *target register*[†] corresponding to the output of the genetic program, and the *operators* connecting and bridging the source and target registers. In VUWLGP, the source and target registers are represented by two floating point (`double`) vectors `cf` and `r`, respectively. The operators can be simple standard arithmetic operators or complex specific functions predefined for a particular task.

An LGP program often has only one register interpreted in determining its output (Oltean *et al.*, 2004; Banzhaf *et al.*, 1998). This kind of form can be easily used for regression and binary classification problems just as in TGP.

In this work, we use LGP for multi-class object recognition problems, where an LGP program is required to produce multiple outputs. Instead of using only one register as the output, we use multiple registers in a genetic program, each corresponding to a single class. An example genetic program for a three-class object classification problem is shown in figure 1(a), where (`cf[0], cf[1], cf[5]`) are the three out of the six image features extracted from the objects and used as source registers, (`r[0], r[1], r[2]`) are the target registers for the class labels, and (`+, -, *`) are the three operators that are automatically selected from the function set during the evolutionary process. The three lines of code evolved starting with `//` are structural *introns*, which do not have direct impact on the output of the program.

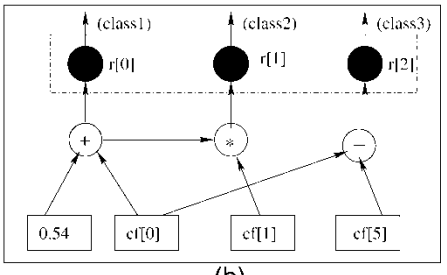
Prior to any program being executed, the registers are zeroed. The features representing the objects to be classified are then loaded into predefined positions in the registers. The program is executed in an imperative manner and can represent a `directed acyclic graph` (DAG), which is different from TGP where a program represents a tree. Any register's value may be used in multiple instructions during the execution of the program. For example, the sample program presented in figure 1(a) can be represented as a DAG, as shown in figure 1(b).

For a program with an object image as input, the class represented by the target register with the largest value is considered the class of the input object image. For an unseen object, if the output target register values are (0.20, 14.92, −3.23), then this object will be classified

[†]There can be more than one target register within a single instruction in some special cases.

```
//[r1] = 0.12 + 0.56;
//r[2] = cf[3] + cf[0]
//r[1] = cf[2] / 3.52;
r[0] = 0.54 + cf[0];
r[1] = r[0] * cf[1];
r[2] = cf[0] - cf[5];
```

(a)



(b)

Figure 1. An LGP example program for multi-class object classification. (a) Example program code; (b) DAG form of the program.

as *class2* (`r[1]` is the largest among the three values for the three registers). A discussion on this concept will be explored in more detail using a real evolved program later in section 5.3.

Note that instructions in an LGP program may also be *introns – i.e.* code whose execution has no impact on the output of the program. While some introns are meaningless for the effectiveness of the system and just slow down the evolution, some others might play a similar role to the automatically defined functions (ADFs) in TGP as 'subroutines' of the program (Koza 1994). This topic will be further investigated later in section 5.4.

In the rest of this section, we describe the genetic operators and the new fitness function developed and used in this approach.

3.2 *Genetic 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.

Two different forms of mutation (Brameier and Banzhaf 1998) were used in this work. *Instruction based mutation* (also known as *Macromutation* previously) replaces an entire instruction with a randomly generated one. *Element based mutation* (also known as *Micro-mutation* previously) changes just one part of an instruction – either the target register, a source register or an operation. These operations can cause dramatic changes in the DAG that a program represents (Brameier and Banzhaf 2001).

In the crossover operator, a part from each of the two parent programs is selected and the two parts are swapped to produce offspring. If a newly produced program is longer than the maximum length allowed, then an instruction is randomly selected and removed. While this is similar to the two-point crossover in genetic algorithms(Goldberg 1989), the two selected parts can have different lengths here.

3.3 *Fitness function*

Since the size of the training set is finite, any fitness function can only be an approximation to a program's true fitness. In many cases, the program's measured fitness does not match the actual performance of the program in relation to the decision boundaries that it is trying to model. In an object recognition problem, a program's true fitness is the 'fraction' of the feature space it can correctly classify. A good fitness function is one that leads to a smooth fitness landscape toward the solution, which is expected to accurately estimate this fraction.

3.3.1 Problems of the old fitness function. A commonly used fitness function for image recognition problem is the *error rate* (or *recognition rate*) of a program recogniser. While it performs reasonably well on some problems, this fitness function frequently fails to accurately estimate the fraction of the feature space correctly classified by a program.

Figure 2(a) shows a simple classification problem with two features `f1` and `f2`. Figure 2(a1) shows the true feature space – feature vectors representing class c1 objects always appear in the fraction of the feature space denoted 'c1', and similarly for the fractions denoted 'c2' and 'c3'. Figure 2(a2) shows that `program1` misclassifies two objects of class c2 as class c3. This program has an error rate of 15% (2/13). Figure 2(a3) shows that `program2` misclassifies one object from class c3 and one object from class c1 as class c2. This program also has an error rate of 15% and will be treated the same as the `program1`. As shown in the two diagrams, `program2` actually classified a greater fraction of the true feature space and approximated the true fitness more accurately than `program1`, but the fitness function cannot accurately reflect this difference and does not distinguish the misclassifications of objects of different classes.

We call this problem *the hurdle problem*, and it usually occurs when (any two) classes have a complex boundary in the feature space, such as those shown in figure 2(b1) and (b2). In such a situation, it is easy to classify the bulk of fitness cases for one class correctly, but learning to classify the other class often initially comes only at an equal or greater loss of accuracy in classifying the first class. Since evolution is short sighted and there is a strong selection pressure against making the feature space classification boundary more complex (a kind of local optimum), GP with such a fitness function often cannot surmount the hurdle within a limited number of generations.

3.3.2 The new fitness function. The hurdle problem can be largely avoided by assuming that a program which correctly classifies objects from more classes is fitter than a program which correctly classifies the same number or slightly more objects but from fewer classes. This assumption can be manifested by a proportionally increased penalty for later misclassifications of the same class than earlier misclassifications. For example, if the first misclassification of a class might attract a penalty of 0.5 and the second might attract a penalty of 1.0, the program diagrammed in figure 2(a2) will attract a penalty of 1.5. The program diagrammed in figure 2(c) will, however, only have a sum penalty of $0.5 + 0.5 = 1.0$ and accordingly has a better fitness. In this way, the fitness function will more accurately model the feature space than the simple error rate. As the evolution goes on, the population's ability to 'jump the hurdle' and accurately model the feature space will grow.

To simulate this idea, we introduced a new fitness function, f, to more accurately measure how well an individual program classifies the feature space. The new fitness function uses

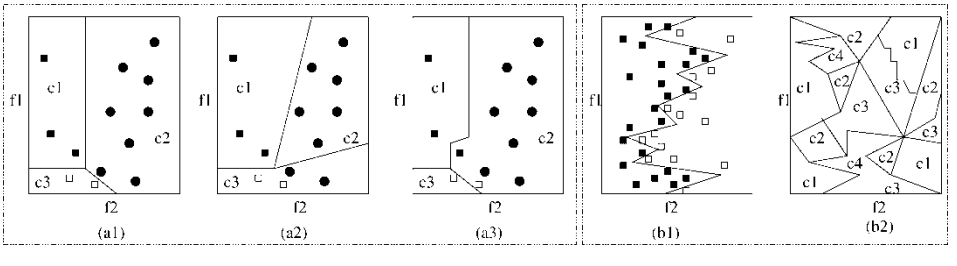


Figure 2. Program fitness *vs* true feature space.

an *increasing penalty* for each of the M_c misclassifications of some class c, as shown in equation 1.

$$f = \frac{1}{N} \sum_{c} \sum_{i=0}^{M_c} \alpha\beta i, \tag{1}$$

where N is the total number of training examples, α and β are constants with $\alpha > 1$ and $\beta > 0$ to guarantee that the penalty of later misclassifications can be exponentially increased. The values of α and β can be determined through empirical search. We used the fitness function to approximate the true fitness so that the penalty of later misclassifications with an $\alpha > 1$ can be exponentially increased.

Obviously, as α approaches 1.0, f becomes more and more similar to the commonly used fitness function (error rate in this case) in object recognition/classification. As β approaches 0.0, the curve becomes progressively flatter. Fitness functions with values of $\alpha < 1$ or with values of $\beta < 0$ are either mathematically nonsensical or exacerbate the hurdle problem by creating a line which slopes the wrong way, which should be avoided in the experiments.

4. Image data sets and experiment configuration

4.1 *Data sets*

Experiments were conducted on three different object data sets providing image classification problems of increasing difficulty. Example images for each data set are shown in figure 3.

The first data set (*shape*, figure 3(a)) 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) formed the data set. The four classes are: dark circles (DC, *class*1), light circles (LC, *class*2), dark squares (DS, *class*3), and light squares (LS, *class*4). Note that the objects between classes C1 and C3 and between classes C2 and C4 are intentionally

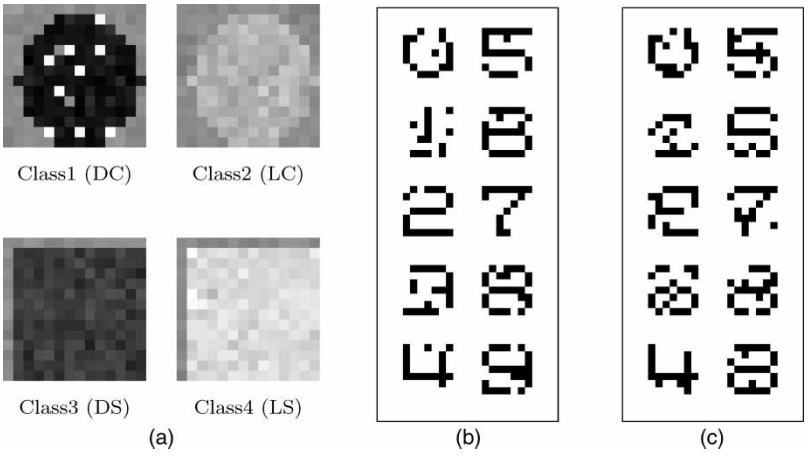


Figure 3. Image data sets. (a) shape; (b) digit1 and (c) digit2.

made very similar in the average value of total pixels, which makes the problem reasonably difficult.

The second and third data sets contain 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 (digits 0, 1, 2, ..., 9) each digit example belongs to. Note that all the digit patterns have been corrupted by noise. In the two tasks (figure 3(b) and (c)), 15% and 30% of pixels, chosen at random, have been flipped. In data set 2 (*digit*1), 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 '9', even '1' and '5'. The task in data set 3 (*digit*2) is even more difficult – human eyes cannot recognise the 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 and the number of features is intentionally made very large (see the next subsection), making the two tasks even more difficult.

For each of the three of these data sets, the training and test sets were 50% of the total data. The training set was used for learning/evolving classifier programs and the test set was used for measuring the system performance.

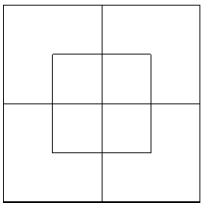
4.2 *Terminal set and function set*

4.2.1 Terminals. In the *shape* data set, we used eight statistical features (`f1, f2, ..., f8` corresponding to the source registers `cf[0], cf[1], ..., cf[7]` in LGP) extracted from different parts of the object images and a random number as the terminal set. The eight features are shown in figure 4.

For the two digit data sets, we used the raw pixels as the terminal sets, meaning that 49 feature terminals were used. The large number of terminals makes the two digit recognition problems more difficult, and we expect the LGP system to automatically select those highly relevant to each recognition problem.

Notice that these features are certainly not the best for solving these particular problems. However, our goal is to investigate the ideas and methods on a linear structure, a fitness function and the program size rather than finding good features for a particular task, which is beyond the scope of this paper.

4.2.2 Functions. The function set for all the three data sets was `{+, -, *, /, if}`. The $+$, $-$, and $*$ operators have their usual meanings – addition, subtraction and multiplication, while / represents 'protected' division which is the usual division operator except that a divide by zero gives a result of zero. Each of these functions takes two arguments. The `if` operator executes the next statement if the first argument is less than the second, and does nothing otherwise.



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

Figure 4. Terminal set for the *shape* data set.

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

Parameters	Shape	Digit15	Digit 30
Pop_size	500	500	500
Max_program_length	16	40	40
Reproduction_rate	10%	10%	10%
Crossover_rate	30%	30%	30%
Instruction_based_mutation_rate	30%	30%	30%
Element_based_mutation_rate	30%	30%	30%
α	1.15	1.15	1.15
β	0.18	0.18	0.18

Again, this set of operators might not be the best set for these particular tasks. However, finding good operators for a specific task is beyond the scope of this paper.

4.3 *Parameters and termination criteria*

The parameter values used for the LGP system for the three data sets are shown in table 1. The evolutionary process is terminated at generation 50 unless a successful solution is found, in which case the evolution is terminated early. These parameter values for the population size, maximum program length, reproduction rate, crossover rate, and the two different kinds of mutation rates were set based on some heuristic guidelines (Banzhaf *et al.*, 1998) and an empirical search through some initial experiments. For the fitness parameters α and β introduced in this paper, we did some initial experiments on these data sets. We found that a value of α between 1.1 and 1.3 and a value of β between 0.15 and 0.40 consistently gave good results. Other values beyond these ranges did not give good (and similar) results. So in our experiments, we used a good combination of values 1.15 and 0.18 for the two parameters, which performed quite well.

4.4 *TGP configuration*

The LGP approach developed in this work will be compared to the TGP approach (Koza 1994). In TGP, the ramped half-and-half method was used for program generation (Banzhaf *et al.* 1998). The proportional selection mechanism and the reproduction, crossover and mutation operators (Koza 1994) were used in evolution. The program output was translated into a class label according to the static range selection method (Loveard and Ciesielski 2001b).

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.

Notice that the program depths above in TGP were derived from the LGP program lengths based on the following heuristic: an LGP instruction typically consists of one or two arguments and an operation, each corresponding to a node in a TGP program tree. Thus an LGP instruction might initially seem to be equivalent to 2–3 nodes. Considering that each TGP operation might be used by its children and parents, an LGP instruction is roughly corresponding to 1.5 tree nodes. Assuming each non-leaf node has two children, we can calculate the capacity of a depth-n TGP from LGP program instructions.

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
Digit1	LGP	$68.62\% \pm 4.67\%$	$65.78\% \pm 5.25\%$
	TGP	$52.60\% \pm 6.65\%$	$51.80\% \pm 6.85\%$
Digit 2	LGP	$55.22\% \pm 3.49\%$	$51.04\% \pm 4.26\%$
	TGP	$41.15\% \pm 5.03\%$	$35.00\% \pm 6.17\%$

5. Results and discussion

5.1 *Object classification performance*

All single experiments were repeated for 50 runs and the mean and standard deviation ($\mu \pm \sigma$) of the results are presented here. Table 2 shows a comparison between the LGP approach developed in this work and the standard TGP approach for the three image recognition problems. Both approaches used the new fitness function proposed in this paper, and this subsection intended to compare the overall accuracy performance of the two approaches. The investigation of the new fitness function performance will be discussed in the next subsection.

On the relatively easy problem in the shape data set, the LGP approach always generated a genetic program which successfully classified all objects in the training set among all the 50 runs. 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. This suggests that the LGP approach greatly outperformed the TGP approach on this data set in terms of the classification accuracy.

On the two difficult digit data sets with a large number of classes and a very high number of dimensions of features, the results show a very similar pattern to those on the shape data set. In both cases, the LGP approach achieved a much higher average value and a lower standard deviation of the classification accuracy on the test set than the TGP approach. On the most difficult digit data set (digit2) in particular, the LGP approach can recognise more than half of the objects in the unseen test set, which is even better than humans.

To see a clear pattern for the improvement, we did a statistical significant test. Since all the 50 runs started with a random seed, we assumed a normal distribution for the results and did a T-test (Berenson and Levine, 1988). In the T-test, a `p-value` less than 0.05 refers to a significant improvement with a confidence level of 95%. The `p-values` of the improvement of the LGP approach over the TGP approach on all the three data sets were all zero, showing that the improvement of the LGP approach on the classification accuracy over the basic TGP approach was *significant* for all these object classification problems.

On all the three data sets investigated here, the improvements of LGP over TGP were more than 10%, which is quite remarkable. Figure 5 shows this pattern even more clearly. This suggests that the LGP approach outperforms the TGP approach for these multi-class object recognition/classification problems.

5.1.1 Discussion on training efficiency. Inspection of the number of generations used reveals that the LGP approach required fewer generations than the TGP approach in finding a good genetic program classifier for these object classification problems. For example, in

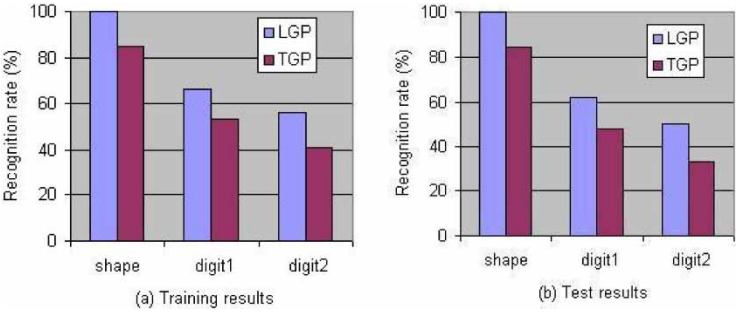


Figure 5. Recognition rates of the LGP and TGP on the three image data sets.

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). This suggests that the LGP approach described in this paper has a stronger evolvability.

5.2 *Impact of the new fitness function*

To investigate whether the new fitness function is helpful in deleting/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 large enough to express a solution to the problem – good solutions have been found even when the maximum length is 5. In TGP the tree depths are left at 3–5. While the lengths normally used in attempting to solve a problem are greater than those allowed here, these limits are likely to be representative of the situation when a much more difficult problem is being addressed. In such problems, the maximum depth which is computationally tractable with existing hardware may also be so short relative to the problem's difficulty where 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 `p-values` on the improvement of the new fitness function over the old one were 0.010 and 0.016 for the training and test accuracy, respectively, showing that 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

Table 3. 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 ± 16.74	77.14 ± 16.68
	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

significant as for the TGP method (the p-values on the improvement were 0.064 and 0.055, which are slightly greater than the common point 0.05). However, this was mainly because the LGP method with the old fitness function already performed quite well (98.76%) due to the power of LGP structure. In addition, when using either the old or the new fitness functions, the LGP method always outperformed the TGP method, which is consistent with our previous observation.

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% $\pm$0.19% on the training set and 74.86% $\pm$ 0.24% on the test set. By using the new fitness function, 36 of the 50 runs finished with test and training accuracies greater than 75%. These results confirmed that the new fitness function performed better than the old one.

5.3 *Analysis of evolved genetic programs*

To check whether the genetic programs evolved by the LGP approach are relatively easy to interpret, we used a typical evolved program which perfectly classified all object images for the shape data set as an example. The core code of the evolved genetic program with structural introns commented using // is shown in figure 6 (left). The graph representation of the program is shown in figure 6 (right) after the introns are removed. In the figure, the filled circles are output class labels corresponding to the target registers in the LGP program, the outlined circles are functions/operators, and the outlined rectangles are referred to as the terminals.

As mentioned earlier, the eight feature terminals (F1...F8) correspond to the source registers (cf[0]...cf[7]), and the target register (r[0]...r[3]) to the four class labels (C1, C2, C3, C4). Given an object to be classified, the target 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 target register values for each object example and make classification for each object image.

Object	Target-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

Accordingly, this program correctly classified all the four object examples. Other evolved programs have a similar pattern to this program. This suggests that the outputs of the programs

```
//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;
```

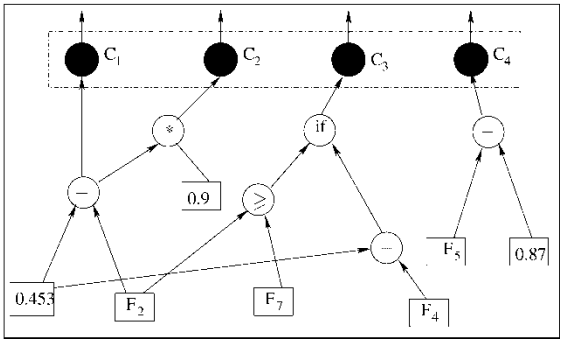


Figure 6. An example evolved LGP program.

```
(* (- (+ (/ F1 -0.268213)
       (/ -0.828695 F6)
    )
    (/ (/ F7 F6)
       (+ -0.828695 F5)
    )
 )
 (* (- (if F3 F1 F5)
       (/ F5 F6)
    )
    (+ (- F4 -0.828695)
       (+ F1 F2)
    )
 )
)
```

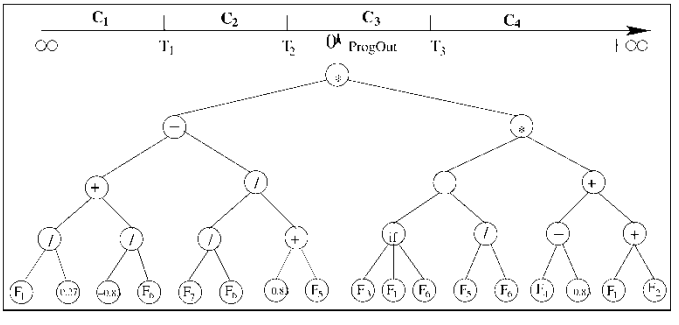


Figure 7. An example evolved TGP program (T_i: the class boundaries; C_i: the class labels).

directly correspond to the multiple classes in the classification problems and the evolved LGP programs are relatively easy to interpret.

Further inspection of this program reveals that only four features were selected from the terminal set and some random numbers were also successfully evolved. This suggests that the LGP approach can automatically select features relevant to a particular task. The graph representation of the program also shows that the LGP approach can co-evolve subprograms together each for a particular class and that some terminals and functions and program fragments can be reused and/or shared by different sub-programs.

An example evolved TGP program is shown in figure 7. The TGP program used almost all the features and only produced a single value (`ProgOut`), which must be translated into a set of class labels but such a translation is often sensitive to the class boundaries. In other words, interpretation of a TGP program for object recognition/classification has to involve some kinds of additional program translation rule, which is a relatively indirect and difficult task for the multi-class object recognition/classification problems.

5.3.1 Further discussion – LGP *vs* TGP for multi-class object classification. According to the *no free lunch* theorem, one approach can hardly do absolutely better than another in terms of all aspects. However, the results shown in the previous two sections clearly show that the LGP approach presented in this paper performed significantly better than the TGP approach on these object classification problems. This is mainly because the proposed LGP approach has a better structure than the TGP approach used here and that the LGP structure is

more suitable than the TGP structure for representing the potential evolved solutions for the multi-class object classification problems.

In the basic TGP approach, the evolved programs only produce one floating point number. For object classification problems, this single number needs to be translated into a set of class labels. In other words, the program output floating number space must be split into a number of regions separated by the class boundaries, each for a particular class. Such an example is shown in figure 7. For any fitness case in the training set, the evolutionary process must evolve an appropriate program whose output can be located in the correct region for the target class of that fitness case. When the approach is to find a good program using selection, crossover and mutation, the search will not only need to look for the program space but also need to consider the restrictions of the different regions in the floating number space for each class. For object classification tasks with a small number of classes such as binary classification with simple class boundaries, this is not a big problem; for the tasks with a large number of classes with complex class boundaries, however, this is clearly more difficult. In addition, this approach also has the following disadvantages. First, the ordering of classes is fixed. For binary classification problems, we only need one boundary value (usually zero) to separate the program output space into two regions, which is quite reasonable. For multiple class problems, fixing the ordering of different classes is clearly not good in many cases, since it is very difficult to set a good ordering of different classes without sufficient prior domain knowledge. Secondly, the subdivision of the real axis (program output space) is also fixed before evolution. It is also very hard to manually determine the appropriate sizes of different regions for hard classification problems without good expertise of a particular problem. In other words, this either requires empirical search through experiments, or needs to evolve these class boundaries during evolution, which put more restrictions into the evolution.

In the LGP approach proposed in this paper, since we used a more natural way – the programs used the multi-outputs (multiple target registers) each for a particular class, the above problem and disadvantages were removed. Neither the output of any target register needs to restricted to any specific region, nor the order of the classes needs to be fixed. Accordingly, this approach is more suitable for the classification tasks with either a large number of classes and/or a complex boundaries between different classes than the TGP approach described in this paper. As the three object classification tasks are exactly of this kind, the proposed LGP approach performed better than the TGP approach on them.

5.4 *Impact of Extra 'Target' Registers*

In TGP, Koza introduced *automatically defined functions* (ADFs) to evolve 'subroutine' structures in GP (Koza 1994). In such a design, the genetic programs and ADFs are evolved together and the programs can take the ADFs as functions or terminals. (Koza 1994) suggested that this approach would achieve better performance and result in more comprehensible programs. In LGP for multi-class image recognition programs, we used the same number of target registers as the number of classes in each program. Also we used *extra target registers* in addition to the target registers to simulate the ADFs in TGP. In this consideration, the original target registers are used to represent the class labels which directly contribute to the classification, while the extra target registers will not have direct relationship with the class labels but only play a role of 'subroutines' which can be 'called' by the partial programs with the original target registers through program construction and genetic operations. In this sub section, we investigate

Table 4. Impact of extra target registers on the shape data set.

Extra registers	Test recognition rate (%)	Extra registers	Test recognition rate (%)
0	**99.84 ± 0.80**		
1	99.65 ± 1.74	6	99.36 ±2.66
2	98.68 ± 6.09	7	96.98 ± 8.66
3	99.31 ± 3.56	8	98.89 ± 4.94
4	99.29 ± 4.10	9	97.97 ± 6.32
5	98.14 ± 6.05	10	95.42 ± 9.37

the impact of the use of extra target registers, which serve as the ADFs, on the system performance.

The results on the *shape* data set using 1–10 extra target registers and same parameter values as in earlier experiments are shown in table 4. According to this table, the use of extra target registers as ADFs in LGP does not improve the object classification accuracy at all. While the use of some extra (*e.g.* 1, 3, 4, 6) registers resulted in similar performance, most extra registers led to clearly worse performance. The results on other data sets also showed a similar pattern, suggesting that the use of extra target registers as ADFs in LGP does not improve system performance as claimed in TGP.[‡]

We believe the major reason is as follows. Firstly, while subprograms in ordinary TGP are very hard to be reused, the use of ADFs in TGP can clearly improve this situation. In LGP, however, due to the linear property, each of the evolved programs actually represents a graph (a DAG), where different parts of the program (subprograms) can be easily reused. The program presented in figure 6 shows such an example. In other words, LGP does not need extra target registers (ADFs) for subroutines and such a structure has already been embedded within its graph structure. Secondly, the use of extra target registers increases the size of the program space, which requires more effort to evolve good programs.

5.5 *Impact of program length*

In TGP and also LGP, there are some existing heuristics for setting program lengths (Banzhaf *et al.*, 1998) based on problem difficulty. However, there has not been any clear guidance for setting program lengths in LGP for multi-class object recognition problems. This subsection investigates this topic by varying the program length parameter ranging from *one fewer instructions than the number of classes to 10 times as many instructions as the number of classes* for the three data sets. The results on the *shape* data set are shown in table 5.

As can be seen from the table, the LGP system with a too small program length cannot achieve good performance. As this length increases to about *four times as many instructions as the number of classes*, the system obtained the best object recognition accuracy. If this length continued to increase, the recognition accuracy will not increase (decrease a bit) but the evolution time was increased (time results are not shown here). The results on the other two data sets showed a similar pattern to the shape set. This suggests that when using LGP for multi-class object recognition, a too small or a too large program length will not lead to the best performance, that there exists *a certain point* (more accurately *a certain range*) which

[‡]Some researchers found that the use of ADFs in TGP does not improve the system performance in some cases (Nanduri and Ciesielski, 2005).

Table 5. Impact of program length on the shape data set.

Program length	Test recognition rate (%)	Program length	Test recognition rate (%)
3	89.46 ± 11.76	12	98.89 ± 4.14
4	88.89 ± 11.64	**16**	$\mathbf{99.73 \pm 0.96}$
5	98.16 ± 5.83	20	99.55 ± 2.61
6	97.33 ± 6.42	25	99.31 ± 3.63
8	97.98 ± 6.33	30	98.67 ± 1.30
10	98.85 ± 5.20	40	97.42 ± 3.53

can maximise the system performance, and that the heuristic *four times as many instructions as the number of classes* can serve as a *starting point* for setting the program length parameter.

Notice that these findings might be problem dependent according to the *no free lunch theorem*. However, given the property of the proposed LGP approach, it would be useful to consider these findings when applying this approach to other similar tasks.

6. Conclusions

The goal of this paper was to investigate an approach to the use of linear structured GP (LGP) and a new fitness function for multi-class object recognition problems. This approach was compared with the basic tree based GP (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 these tasks in terms of classification accuracy and training generations.

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

Inspection of the evolved genetic programs reveals that the program classifiers evolved by the LGP approach are relatively easy to interpret for these problems. The results and analyses 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 co-evolve subprograms together each for a particular class in multi-class object recognition problems.

The use of extra target registers as ADFs in LGP does not improve the system performance for these problems. When using LGP for multi-class object recognition, a too small or too large program length will not result in the best performance. The heuristic *four times as many instructions as the number of classes* can serve as a starting point for setting the program length parameter for similar tasks.

Although developed for object recognition problems, we expect that this approach can be applied to general multi-class classification problems such as those in the UCI Machine Learning repository, but this needs to be investigated in the future.[§]

[§]In fact, we also did experiments on three subsets of the Yale Faces Database B (Georghiades *et al.*, 2001), and the results showed a similar pattern.

In the proposed LGP approach, we used multiple target registers in the linear structure as a whole for the multi-class object classification problems, which outperformed the TGP approach described in this paper on these problems. In the future, it would be interesting to investigate the use of the LGP approach with a single target register as output of the program to reveal whether the most important contributor to this improved performance is the linearity of program components or the multiple output value structure.

Acknowledgement

We would like to thank the anonymous referees for their time, comments and suggestions which provide great help for improving the paper.

This work was supported in part by the Marsden Fund Council from the government funding (05-VUW-017), administrated by the Royal Society of New Zealand and the University Research Fund (7/39) at Victoria University of Wellington.

References

D. Andre, "Automatically defined features: The simultaneous evolution of 2-dimensional feature detectors and an algorithm for using them", in *Advances in Genetic Programming*, K.E. Kinnear, Ed., 1994, Cambridge, MA: MIT Press,
pp. 477–494.

M.R. Azimi-Sadjadi, D. Yao, Q. Huang and G.J. Dobeck, "Underwater target classification using wavelet packets and neural networks", *IEEE Trans. Neural Netw.*, 11, pp. 784–794, 2000.

J. Bala, K.D. Jong, J. Huang, H. Vafaie and H. Wechsler, "Using learning to facilitate the evolution of features for recognising visual concepts", *Evol. Comput.*, 4, pp. 297–312, 1997.

W. Banzhaf, P. Nordin, R.E. Keller and F.D. Francone, *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann: dpunkt.verlag, 1998.

M.L. Berenson and D.M. Levine, *Applied statistics: a first course*, NJ, USA: Prentice-Hall, Inc., Upper Saddle River, 1998.

B. Bhanu, Y. Lin and K. Krawiec, *Evolutionary Synthesis of Pattern Recognition Systems*, Monographs in Computer Science. New York: Springer-Verlag, 2005.

M. Brameier and W. Banzhaf, "A comparison of Genetic Programming and neural networks in medical data analysis". Reihe CI 43/98, SFB 531, Dortmund University, Germany (1998).

M. Brameier and W. Banzhaf, "Effective linear Genetic Programming". Technical report, Department of Computer Science, University of Dortmund, 44221 Dortmund, Germany (2001).

T. Caelli and W.F. Bischof, *Machine Learning and Image Interpretation*, New York and London: Plenum Press, 1997, ISBN 0-306-45761-X.

S. Cagnoni, E. Lutton and G. Olague, *Genetic and Evolutionary Computation for Image Processing and Analysis, EURASIP Book Series on Signal Processing and Communications*. Hindawi Publishing Corporation (in press) 2007.

V. Ciesielski, A. Innes, S. John and J. Mamutil, "Understanding evolved genetic programs for a real world object detection problem", in *Proceedings of the 8th European Conference on Genetic Programming*, vol. 3447 of *Lecture Notes in Computer Science*, M. Keijzer, A. Tettamanzi, P. Collet, J.I. van Hemert and M. Tomassini, Ed., Lausanne, Switzerland: Springer, 2005, pp. 351–360.

D. de Ridder, A. Hoekstra and R.P.W. Duin, "Feature extraction in shared weights neural networks", in *Proceedings of the Second Annual Conference of the Advanced School for Computing and imaging, ASCI*, Delft, 1996, pp. 289–294.

M.H. Dunham, *Data Mining: Introductory and Advanced Topics*, New Jersey: Prentice Hall, 2003.

C. Fogelberg and M. Zhang, "Vuwlgp – an ansi c++ linear Genetic Programming package". Technical Report, School of Mathematics, Statistics and Computer Science, Victoria University of Wellington (2005).

D.A. Forsyth, and J. Ponce, *Computer Vision: A Modern Approach*, New Jersey: Prentice Hall, 2003.

A. Georghiades, P. Belhumeur and D. Kriegman "From few to many: Illumination cone models for face recognition under variable lighting and pose", *IEEE Trans. Pattern Anal. Mach. Intell.*, 23, pp. 643–660, 2001.

D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading, MA: Addison–Wesley, 1989.

H. Gray, "Genetic Programming for classification of medical data"in *Late Breaking Papers at the 1997 Genetic Programming Conference*, J.R. Koza, Ed., California: Stanford University, 1997, pp. 291–297.

D. Howard, S.C. Roberts and R. Brankin, "Target detection in SAR imagery by Genetic Programming", *Adv. Eng. Softw.*, 30, pp. 303–311, 1999.

D. Howard, S.C. Roberts and C. Ryan, "The boru data crawler for object detection tasks in machine vision", in *Applications of Evolutionary Computing, Proceedings of EvoWorkshops2002: EvoCOP, EvoIASP, EvoSTim*, Vol. 2279 of *LNCS*, S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf and G. Raidl, Ed., Kinsale, Ireland: Springer-Verlag, 2002, pp. 220–230.

J.K. Kishore, L.M. Patnaik, V. Mani and V.K. Agrawal, Application of Genetic Programming for multicategory pattern classification. *IEEE Trans. Evol. Comput.*, 4, pp. 242–258, 2000.

J.R. Koza, *Genetic Programming*, Cambridge, Massachusetts: MIT Press, 1992.

J.R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*, Cambridge, Mass, London, England: MIT Press, 1994.

K. Krawiec and B. Bhanu, "Visual learning by coevolutionary feature synthesis", *IEEE Trans. Syst., Man, and Cybern. B*, 35, pp. 409–425, 2005.

Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition", in *Intelligent Signal Processing*, Chichester, UK: IEEE Press, 2001, pp. 306–351.

T. Loveard, and V. Ciesielski, "Representing classification problems in Genetic Programming", in *Proceedings of the Congress on Evolutionary Computation*, Vol. 2, pp. 1070–1077, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea: IEEE Press, 2001a.

T. Loveard and V. Ciesielski, "Representing classification problems in Genetic Programming", in *Proceedings of the Congress on Evolutionary Computation*, Vol. 2, pp. 1070–1077, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea: IEEE Press, 2001b.

D.P. Muni, N.R. Pal and J. Das, "A novel approach to design classifier using Genetic Programming", *IEEE Trans. Evol. Comput.*, 8, pp. 183–196, 2004.

D.T. Nanduri and V. Ciesielski, Comparison of the effectiveness of decimation and automatically defined functions, *KES'OS, Lecture Notes in Artificial Intelligence*, 3683, pp. 540–546, 2005.

G. Olague, S. Cagnoni and E. Lutton, (Eds) "Special issue on evolutionary computer vision and image understanding", *Pattern Recognit. Lett.* 27, 2006.

M. Oltean, C. Grosan and M. Oltean, Encoding multiple solutions in a linear Genetic Programming chromosome", in *Computational Science - ICCS 2004: 4th International Conference, Part III*, volume 3038 of *Lecture Notes in Computer Science*, M. Bubak, G.D. van Albada, P.M.A. Sloot and J. Dongarra, Ed., Krakow, Poland: Springer-Verlag, 2004, pp. 1281–1288.

T.K. Paul and H. Iba, "Identification of weak motifs in multiple biological sequences using genetic algorithm", in *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, New York, NY, USA: ACM Press, 2006, pp. 271–278.

G. Robinson and P. McIlroy, "Exploring some commercial applications of Genetic Programming", in *Evolutionary Computation, Vol. 993*, T.C. Fogarty, Ed., *Lecture Note in Computer Science*. Berlin: Springer-Verlag, 1995.

S. Russell and P. Norvig, *Artificial Intelligence, A Modern Approach*, New Jersey: Prentice Hall, 2nd Ed., 2003.

W. Smart, and M. Zhang, "Classification strategies for image classification in genetic programming", in *Proceeding of Image and Vision Computing Conference*, D. Bailey, Ed., Palmerston North, New Zealand, 2003, pp. 402–407.

W. Smart and M. Zhang, "Using Genetic Programming for multiclass classification by simultaneously solving component binary classification problems", in *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, M. Keijzer, A. Tettamanzi, P. Collet, J.I. van Hemert and M. Tomassini, Ed., Lausanne, Switzerland: Springer, 2005, pp. 227–239.

A. Song, "*Texture Classification: A Genetic Programming Approach*". PhD thesis, Department of Computer Science, RMIT University, Melbourne, Australia (2003).

A. Song and V. Ciesielski, "Texture analysis by Genetic Programming", in *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pp. 2092–2099, Portland, Oregon: IEEE Press, (2004).

W.A. Tackett, "Genetic Programming for feature discovery and image discrimination", in *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, S. Forrest, Ed., University of Illinois at Urbana-Champaign: Morgan Kaufmann, 1993, pp. 303–309.

W.A. Tackett, *Recombination, Selection, and the Genetic Construction of Computer Programs*. PhD thesis, Faculty of the Graduate School, University of Southern California, Canoga Park, California, USA (1994).

A. Teller and M. Veloso, "A controlled experiment : Evolution for learning difficult image classification", in *Proceedings of the 7th Portuguese Conference on Artificial Intelligence*, volume 990 of *LNAI*, C. Pinto-Ferreira and N.J. Mamede, Ed., Berlin: Springer Verlag, 1995a, pp. 165–176.

A. Teller and M. Veloso, "PADO: Learning tree structured algorithms for orchestration into an object recognition system", Technical Report CMU-CS-95-101, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA (1995b).

B. Verma, "A neural network based technique to locate and classify microcalcifications in digital mammograms", in *1998 IEEE World Congress on Computational Intelligence – IJCNN'98*, Anchorage, Alaska. 0-7803-4859-1/98, IEEE, 1998, pp. 1790–1793.

J.F. Winkeler and B.S. Manjunath, "Genetic Programming for object detection", in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, J.R. Koza, K. Deb, M. Dorigo, D.B. Fogel, M. Garzon, H. Iba and R.L. Riolo, Ed., Stanford University, CA, USA: Morgan Kaufmann, 1997, pp. 330–335.

P. Winter, W. Yang, S. Sokhansanj and H. Wood, "Discrimination of hard-to-pop popcorn kernels by machine vision and neural network", in *ASAE/CSAE meeting*, Saskatoon, Canada. Paper No. MANSASK 96-107, 1996.

Y.C. Wong and M.K. Sundareshan, "Data fusion and tracking of complex target maneuvers with a simplex-trained neural network-based architecture", in *1998 IEEE World Congress on Computational Intelligence – IJCNN'98*, Anchorage, Alaska, 0-7803-4859-1/98, 1998, pp. 1024–1029.

X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks", *IEEE Transactions on Neural Netw.*, 8, pp. 694–713, 1997.

M. Zhang, V. Ciesielski and P. Andreae, "A domain independent window-approach to multiclass object detection using Genetic Programming", *EURASIP J. Signal Process., Special Issue on Genetic and Evolutionary Computation for Signal Processing and Image Analysis*, 2003, pp. 841–859, 2003.

M. Zhang and W. Smart, "Multiclass object classification using Genetic Programming", in *Applications of Evolutionary Computing, EvoWorkshops2004: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, EvoSTOC*, volume 3005 of *LNCS*, G.R. Raidl, S. Cagnoni, J. Branke, D.W. Corne, R. Drechsler, Y. Jin, C. Johnson, P. Machado, E. Marchiori, F. Rothlauf, G.D. Smith and G. Squillero, Ed., Coimbra, Portugal: Springer Verlag, 2004, pp. 369–378.

M. Zhang and W. Smart, "Using Gaussian distribution to construct fitness functions in genetic programming for multiclass object classification", *Pattern Recognit. Lett.*, 27, pp. 1266–1274, 2006. Evolutionary Computer Vision and Image Understanding.