New Crossover Operators in Linear Genetic Programming for Multiclass Object Classification

Carlton Downey
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
Carlton.Downey@ecs.vuw.ac.nz

Mengjie Zhang
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
Mengjie.Zhang@ecs.vuw.ac.nz

Will N Browne
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
Will.Browne@ecs.vuw.ac.nz

ABSTRACT

Genetic programming (GP) has been successfully applied to solving multiclass classification problems, but the performance of GP classifiers still lags behind that of alternative techniques. This paper investigates an alternative form of GP, Linear GP (LGP), which demonstrates great promise as a classifier as the division of classes is inherent in this technique. By combining biological inspiration with detailed knowledge of program structure two new crossover operators that significantly improve performance are developed. The first is a new crossover operator that mimics biological crossover between alleles, which helps reduce the disruptive effect on building blocks of information. The second is an extension of the first where a heuristic is used to predict offspring fitness guiding search to promising solutions.

Categories and Subject Descriptors

1.2.2 [**Artificial Intelligence**]: Automatic Programming; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Algorithms, Design

Keywords

Genetic programming, crossover operator, classification

1. INTRODUCTION

Identifying and classifying objects is a common and important task which humans perform daily. Distinguishing between a one and a zero, identifying a healthy person from one with swine flu, and detecting potential terrorists from fingerprint images are just three examples of important object classification tasks. While some classification tasks are trivial for humans, many others are challenging or tedious, encouraging researchers to develop automated classifiers. In many cases, performing these tasks by human/hand is too expensive or too slow. Given the amount of image data containing important objects that need to be detected and classified, computer based solutions to many of these tasks would be of immense social and economic value.

Derived from genetic algorithms [6], Genetic Programming (GP) [1, 9] is a promising and nature inspired approach for constructing reliable classification programs quickly and automatically, given only a set of object image instances on which an evolved 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. Since the 1990s, GP has been successful for solving many object classification problems [10, 16, 26, 27, 29].

The object classification problems that require distinguishing between objects of more than two types are known as multiclass classification problems. Many important classification tasks such as digit recognition are examples of multiclass classification problems. Unfortunately the conventional form of GP, Tree-based GP (TGP), often performs poorly as a classifier for multiclass problems [28].

In order to improve GP performance on multiclass problems a different form of GP, Linear GP (LGP), where programs are represented as a linear sequence of instructions, is considered [2]. It has been demonstrated that LGP has significantly superior performance to TGP on many multiclass classification problems [4, 15]. However, despite many such promising initial results, it appears that relatively scant research has been done in the area of LGP for multiclass classification. The motivation is to improve the performance of LGP as we believe it has potential as a multiclass classifier, which has not yet been fully realised.

The success of GP as a technique has been attributed in large part to the crossover operator; crossover theoretically allows two programs with disparate strengths to produce an offspring that possesses the combined strengths of both parents. Because of this many improved crossover operators have been developed, however such developments have focused almost exclusively on TGP [5, 11, 17, 27]. LGP algorithms differ from TGP algorithms in many important characteristics, such as linear versus tree structure of the learnt program and separable blocks of code versus complete trees. Thus, it is unlikely a crossover operator developed specifically for TGP will perform well when using LGP. Hence it is

GECCO'10, July 7–11, 2010, Portland, Oregon, USA.

considered to improve LGP by developing a new crossover operator that makes the most of the special characteristics of LGP.

Crossover in LGP currently still has two major problems, which are open to attack. It has been shown that crossover often disrupts good building blocks of information instead of preserving them. It is clearly desirable that we restrict the crossover operator in such a way that it minimises building block disruption. It is noted that some disruption may be unavoidable in crossover and may even be necessary to escape from local optima.

There has been a great deal of research into better crossover operators, where the likelihood of good offspring being produced is increased. An example of this is brood recombination crossover [22, 23], where many offspring are produced, with the best kept and the rest discarded. Unfortunately, a large proportion of these improved operators require extra program fitness evaluations, which greatly slows down the evolutionary process. In addition, the number of possible offspring resulting from any given crossover is exceedingly large, while the number of possible good offspring is very small. This results in any crossover method that requires a generate-and-test methodology, such as that used in brood crossover, to be an inefficient solution. What is required is a crossover method that can predict which offspring will be good without actually having to try them.

1.1 Objectives

In this paper, we aim to use the special characteristics of LGP to develop two new crossover operators, which at least partially overcomes the above problems. Specifically, this paper has the following research objectives:

- To determine an abstract structure for LGP programs, and use this structure to develop a new crossover operator that alleviates the problem of building block disruption.
- To improve this new crossover operator by devising a heuristic that predicts which parts of the parent programs should be exchanged in order to maximize the probability of high fitness offspring.
- To compare the performance of LGP with the new crossover operators to that of LGP using the conventional crossover operator.

1.2 Organisation

The rest of the paper is organised as follows. Section 2 describes some necessary background related to GP for object classification, LGP program structure and the standard LGP crossover operator. Sections 3 first analyses the building blocks in LGP then describes the first new crossover operator, *class graph crossover*, developed in this paper. Section 4 describes the second new LGP crossover operator, *selective crossover*. The experiment design and configurations are provided in section 5 and the results are presented in section 6 with discussions. Section 7 concludes the paper and gives future work directions.

2. BACKGROUND

2.1 GP Related Work to Object Classification

Since the early 1990s, a variety of GP systems with different representations have been developed. These include tree-based GP [9], linear GP [2], graph-based GP [2], linear-graph GP [8], grammar-based GP [24], and even machine code instruction based GP [14]. Among them, tree-based or tree-like GP is the most commonly used representation and has been successfully applied to a range of real world classification problems [10, 12, 16, 21, 26, 27, 29], demonstrating the potential of GP as a general method to solve classification problems.

In the first few years after GP was introduced, the classification tasks that the tree-based GP was used to solve were mainly binary classification with two classes only. Since the late 1990s, GP has also been applied to multi-class classification with three or more classes. In the last ten years, there have been several major dimensions of developments, to be summarised below, in tree-based GP for classification.

One major aspect is the investigation of ways of translating the single program output value into a small set of class labels. For binary classification problems, there is a natural translation of negative values to one class and positive values to the other class. For multi-class classification problems, finding the appropriate region boundaries on the numeric value to distinctly separate the different classes is more difficult. Zhang and Ciesielski developed an algorithm [25] that uses a fixed translation into the class labels, which has been used in a number object classification tasks [20, 18, 12] and achieved reasonable classification results on relatively easy classification problems. While it is perhaps the first approach to multi-class classification using a single genetic program, this method usually needs hand crafting of good class boundaries before evolution, which is very hard to do without domain expertise. To avoid this problem, dynamic range selection [13], and centred and slotted dynamic class boundary determination methods [28] were developed. A probability based class translation rule [29] greatly improves GP capability for classification tasks with a relatively small number of classes.

In terms of program representations and structures for classification, there have been several developments. These include the use of multiple programs each for a particular class [10] and each for a binary classification [13] with multiple independent runs, and recent developments that allow multiple programs within a population to be evolved together within a single run [29]. 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 problem.

In the approach to using a single genetic program for the entire multi-class problem, the single program will need to have a directly corresponding relationship 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 for object classification in the unseen test set is often better than the multi-program approach. Another advantage is that the conversion from multi-class to binary class is removed. However, a major disadvantage of this approach is that a single output from the root node in the basic TGP must be translated into a set of class labels. This disadvantage can be avoided in LGP

where multiple registers are evolved concurrently. This paper will be focused on LGP.

2.2 LGP Program Structure and Crossover

The LGP used in this paper follows the ideas of the register machine LGP [1]. In the LGP 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 by the system, a *destination register*, corresponding to the output of the genetic program, and the *operators* connecting and bridging the source and destination registers. For presentation convenience, the destination registers are represented by a floating point (`double`) vector `r`, and the source registers are presented by $\mathtt{f}_i$ denoting the ith feature for the task. 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 [1]. 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 program, each corresponding to a particular class. In other words every LGP program is a complete classifier.

For a program with an object image as input, the class represented by the destination register with the largest value is considered the class of the input object image. For an unseen object in a three-class problem, if the output destination register values are (0.20, 14.92, -3.23), then this object will be classified as *class2* as the middle value is the largest among the three values.

An example LGP program is shown in figure 1. Note that some instructions in an LGP program may also be *introns*, i.e. code whose execution has no impact on the output of the program.

```
r[1] = 3.1 + f1;
r[3] = f2  / r[1];
r[2] = r[1] * r[1];
r[1] = f1  - f1;
r[1] = r[1] - 1.5;
r[2] = r[2] + r[1];
```

Figure 1: An example LGP program.

In the LGP canonical/standard crossover operator, a part consisting of one or more instructions is selected from each of the two parent programs and the two parts are swapped to produce two new offspring.

3. CLASS GRAPH CROSSOVER

3.1 Building Blocks in LGP

A crossover is considered destructive if one or more building blocks are disrupted by the exchange of genetic material. A building block is disrupted if part of it is selected in the code to be exchanged, and part of it is not. Hence non-destructive crossover exchanges only entire building blocks, which is not the case with conventional crossover.

The conventional crossover operator selects the instruction(s) to exchange at random. This means that whether or not an entire building block is selected for exchange is entirely random. If x instructions are selected at random for crossover out of a program with n instructions, then the probability that a building block of size b is disrupted can be calculated.

A building block is not disrupted if either it is entirely exchanged or no part of it is exchanged.

- Number of possible code segments that could be exchanged = C_x^n.[1]
- Number of possible exchanges containing entire building block = C_{x-b}^{n-b}.
- Number of possible exchanges disjoint from building block = C_x^{n-b}.
- Prob of no disruption = $(C_{x-b}^{n-b} + C_x^{n-b})/C_x^n$.
- Prob of disruption = $1 - (C_{x-b}^{n-b} + C_x^{n-b})/C_x^n$.

As the numerator $(C_x^{n-b} + C_{x-b}^n)$ increases, the probability of disruption decreases. Assuming $x \leq n/2$, which is standard practice, the numerator increases as b (the size of the building block) decreases. In other words, as building block size increases, the likelihood the building block is disrupted by crossover also increases. Certainly, once building blocks exceed a very minimal size, the likelihood they will be disrupted by crossover becomes overwhelming. This is an undesirable situation, as this effectively limits the size of building blocks in the population. Successfully solving a problem usually requires constructing larger building blocks from small ones, but this process is often retarded by conventional crossover. It is desired to create a crossover operator that has a less disruptive influence on larger building blocks.

3.2 Biological Crossover

In biology, crossover is limited to exchange of paired DNA [7]. Each physical feature (e.g. eye colour) has many different DNA sequences, called alleles, which code for variations of that feature. When crossover occurs, the alleles of one parent that code for certain physical features are exchanged with the alleles of another parent that code for the same physical features. In other words, biological crossover always exchanges DNA codes for different variations of the same features. For instance the DNA for blue eye colour might be exchanged with the DNA for brown eye colour, but the DNA for blue eye colour would not be exchanged with the DNA for brown hair colour. If we consider the biological building blocks as DNA sequences within a single allele, then biological crossover never destroys building blocks.[2] Alleles are either exchanged in entirety or left alone, hence allele building blocks are also either entirely exchanged, or entirely left alone.

In the remainder of this paper, we will refer to two sequences of code, DNA or program, as *position equivalent* if they code for different versions of the same feature, in other words if they are alleles for the same feature. For example, two alleles for eye colour are position equivalent.

[1] $C_x^n = \binom{n}{x} = \frac{n!}{x!(n-x)!}$

[2] However, some building blocks address epistasis, where the value of one allele affects the importance of another allele [3], so these may still be disrupted.

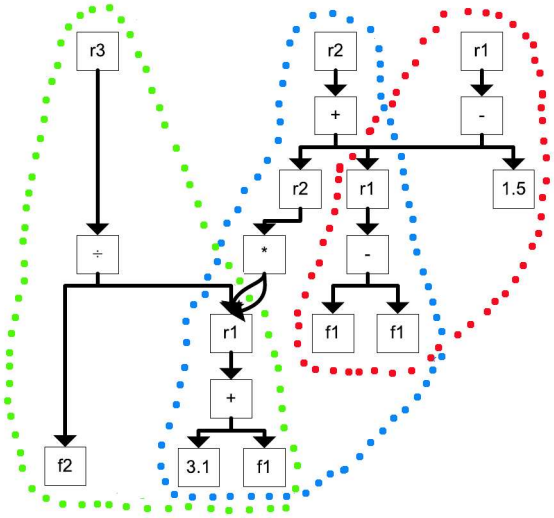


Figure 2: DAG with the 3 class graphs marked.

3.3 Class Graphs

Biological crossover is in stark contrast to conventional GP crossover, where the program code to be exchanged is chosen at random. By ignoring the abstract structure of the program, no distinction for building block boundaries is made. Thus conventional crossover is highly likely to be a destructive influence on building blocks. Taking inspiration from biological crossover, in order to preserve this structure when selecting the code to exchange, an abstract structure within our programs similar to the alleles of biology is defined. Note that a common use of the term allele in evolutionary computation indicates a feature corresponding to a single input variable, whereas here a single feature may correspond to several input variables.

In LGP for multiclass classification there is a natural structure already imposed on programs. An LGP solution to an n-class classification problem can be viewed as a Directed Acyclic Graph (DAG) composing of n overlapping graphs. We define the *nth class graph* in any program as the subgraph whose root is the nth register. In other words, the graph associated with the class n. The DAG format of the example LGP program (figure 1) is given in Figure 2, where the class graphs are indicated. Therefore sequences of instructions are position equivalent if and only if they are class graphs for the same feature/class. This leads us naturally to develop a new crossover operator based on the exchange of position equivalent class graphs. Henceforth this new form of crossover will be known as *Class Graph Crossover (CGC)*.

3.4 Determining Instructions for Class Graphs

To perform class graph crossover, we need to determine the instructions for each class graph in a program. The instructions belonging to every class graph in a problem can be successfully determined by a single backwards pass through the program.

- The classes whose class graphs will be exchanged are selected at random and these class indices are added to a set S.
- Each instruction i is selected in turn by iterating through the instructions in the program in a reverse order (from the last instruction to the first instruction).
  - Select instruction i. If the destination register r_i is in S, then r_i is part of a class graph being exchanged. We add i to the list of instructions to exchange and remove r_i from S. Since the value of r_i depends on its arguments, we then add all arguments of r_i that are registers (i.e. not constants or features) to S.

Comparing the transition of LGP program (figure 1) to DAG (figure 2) with the crossover point selection (figure 3) facilitates the understanding of this procedure used in the class graph identification.

3.5 Further Discussion

3.5.1 Building Blocks

In CGC, a random number of classes are selected to undergo crossover. If a building block is present in the program, it must affect the final value of a class register. This means it is entirely contained within the class graph of that class. Hence, if that class graph is exchanged, the entire building block is exchanged as well. Since CGC swaps entire class graphs, it exchanges nothing but entire building blocks. This is a distinct improvement on conventional crossover where partial building blocks may be exchanged.

While CGC ensures that only entire building blocks are exchanged, we have said nothing of those building blocks not exchanged. Because LGP programs consist of overlapping class graphs, it is possible that part of a building block may belong to more than one class graph. This means building blocks not exchanged may be disrupted by CGC. However, CGC replaces every class graph with a position equivalent class graph, and position equivalent class graphs code for the same class. Two code sequences that aim to perform the same function are statistically more likely to be similar than two randomly selected instruction sequences. Consequently, position equivalent class graphs are more likely to have similar code than two random sequences of instructions. It is possible that disrupted building blocks may be repaired by the code that replaces the amputated part of the building block. Because position equivalent class graphs are more likely to have similar code, the likelihood of repair is higher under CGC. So even though building blocks that are not exchanged may be disrupted by CGC, they are more likely to be repaired by the substituted code.

3.5.2 Crossover Complexity

The difference between CGC and conventional crossover becomes most apparent if the number of possible crossovers which can occur under each technique is considered.

Under a conventional crossover scheme, instructions are exchanged at random. Assume for simplicity's sake that the number of instructions exchanged is the same for both parents, i.e. x instructions from program 1 are exchanged with x instructions from program 2. If x instructions from programs of size n are exchanged, then:

- Number of possible sets of instructions from a single program $= C_x^n$.
- Number of possible crossovers $= C_x^n \times C_x^n$.

Under a CGC scheme, we exchange only class graphs during crossover. Hence if we exchange y class graphs for a problem with c classes:

- Number of possible crossovers $= C_y^c$.

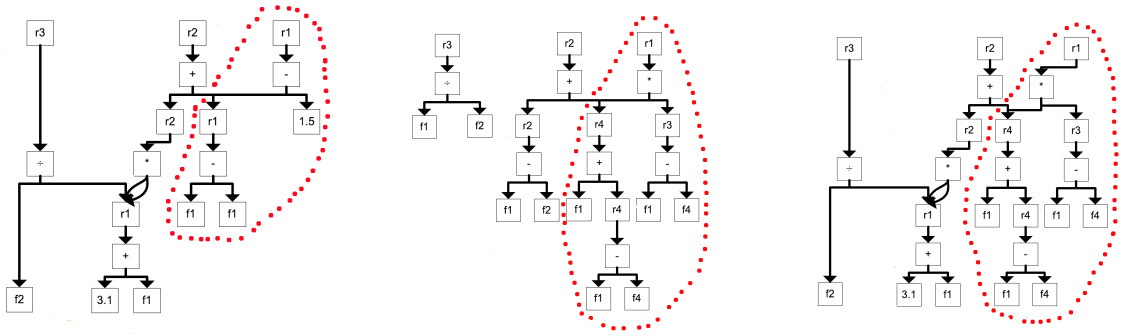


Figure 3: Example of Class Graph Crossover. The r1 class graph is exchanged. Crossover point occurs at the edge of class graph.

Note that the number of possible crossovers depends only on the number of classes and the number of classes to be exchanged, i.e. it is *independent of the program size*. To investigate the effects of this property an example based on the very modest maximum program size used during these experiments is given below:

- Max program size = 32, number of classes = 10.
- Let crossover be 50% of program size, so 16 instructions or 5 class graphs.
- Conventional Crossover = $C_{16}^{32} \times C_{16}^{32} = 3.73 \times 10^{17}$
- Class Graph Crossover = $C_5^{10} = 2.52 \times 10^2$

CGC has reduced the number of possible crossovers by *15 orders of magnitude*, even for this very modest program size. This indicates how severely the CGC operator restricts code exchange compared with conventional crossover.

The purpose of crossover is to search for the code to exchange that results in the best children, with the conventional crossover operator doing this by blind trial and error. The problem complexity is proportional to the number of possible children, and hence the number of possible crossovers. Thus if the problem complexity can be decreased by many orders of magnitude and achieve at least similar results, there will be efficiency improvements in the crossover operator. Further improvements can be obtained if we can move away from trial and error, and instead investigate solutions in an intelligent order. This idea, in the form of a new crossover operator, which will be referred to as *Selective Crossover (SC)* is the focus of the next section.

4. SELECTIVE CROSSOVER

When improving the crossover operator, the goal is to increase the fitness of the offspring produced. As discussed in the introduction, one approach would be to generate and test some number of possible offspring, and discard all but the best, however this is a poor solution as it has huge computational overhead. It is required to determine a heuristic that can be used to *predict* which instructions should be exchanged in order to produce the optimal offspring.

4.1 Motivation for Selective Crossover

By definition, each class graph is associated with a class c, and hence each class graph is responsible for all of the misclassified training instances of class c. Good class graphs are responsible for a low number of misclassifications; bad class graphs are responsible for a large number of misclassifications. A good program has good fitness, and hence a small number of misclassifications. Hence all of the class graphs in a good program are good class graphs, i.e. each class graph is responsible for only a small number of misclassifications. In summary, the aim is a program that consists entirely of good class graphs. Replacing a poor class graph with a good class graph is hypothesised as more likely to improve program fitness than replacing a random class graph with another random class graph. This is the motivation for this novel selective crossover operator.

4.2 One Child Crossover

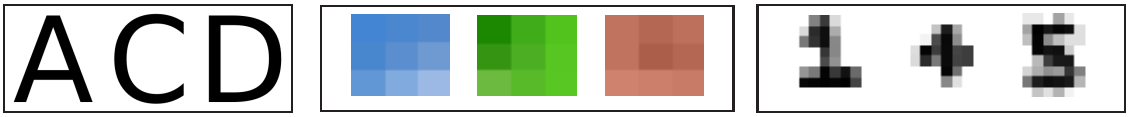
Normal crossover takes two parents and produces two children by an exchange of code, however this has a major issue if heuristics are utilised in optimising the offspring produced. In order to optimize one child, it will receive all of the good code identified by the heuristic. This means the second offspring will receive the remaining, rejected code. Hence it is likely that the more optimized one offspring, the relatively worse the second child. Hence for the remainder of this paper only the one 'good' child will be produced.

4.3 Algorithm

Because the correctness of each class graph is represented by a single integer, the difference in correctness between class graphs can be calculated by simple subtraction. The class graphs are then ranked in descending order so that exchanging x class graphs is achieved by substituting the first x class graphs in the ordered list. In other words, the algorithm determines the class graphs in the second program that are superior to the position equivalent class graphs in the current program, and exchanges them. So suppose parent one is poor at classifying certain class instances, whereas parent two is good at classifying these same class instances. Then the selective crossover operator would be expected to replace the poor code in program one with the good code from program two.

- Count the number of misclassified training instances for each class in both parents.
- Order the class graphs by the difference.
- Choose some random number x.
- Select the first x class graphs from the ordered list.
- Exchange these class graphs.

Notice that the number of misclassification count here only considers "false negatives", namely the miss instances



(a) Artificial Characters (b) Image Segmentation (c) Handwritten Digits

Figure 4: Examples of instances in the image data sets.

for a particular class. For example, for class C the number of missclassifications would be the total number of objects of class C that were incorrectly classified by the program as some other classes. In this way, the sum of the misclassifications for all classes will be equal to the total number of incorrectly classified objects across all classes, that is, the error rate for the entire multiclass object classification problem. This is to be consistent with the TGP used in this paper, where the error rate was used as the fitness function. However, considering false positives simultaneously might improve the system performance, but this needs to be further investigated in the future.

5. EXPERIMENTAL CONFIGURATIONS

In order to compare the effectiveness of different GP methods as techniques for performing multiclass classification, a series of experiments was conducted in the important multiclass problem domain of object recognition. Here we describe the image data sets and parameters used during the course of these experiments.

5.1 Image Data Sets

We used three image data sets providing object classification problems of varying difficulty in the experiments. These data sets were chosen from the UCI machine learning repository [19]. Example instances of all three data sets can be found in figure 4.

The first data set is *Artificial Characters*. This data set consists of vector representations of 10 capital letters from the English language. It has a high number of classes (10) and a high number of attributes/features (56), and 5000 instances in total. The second set is *Image Segmentation*, which consists of 3x3 regions drawn from images of outdoor areas such as sky, grass etc. It has 7 classes, 19 attributes and 2100 instances. The third set is *Hand Written Digits*, which consists of 3750 hand written digits with added noise. It has 10 classes and 564 attributes. These tasks are quite hard due to a high number of attributes, a high number of classes and noise in some sets.

5.2 TGP/LGP Parameters

These values were determined based on common settings for different LGP methods and empirical search via initial experiments, which are shown in table 1. TGP used the same settings except that the program depth was set to 8, corresponding to 32 instructions in LGP. For all experiments, initial programs in the population consist of randomly chosen instructions.

5.3 Experimental Configurations

Each data set was divided into a test set, a training set, and a validation set with all sets being of equal size and classes. After the maximum number of generations was reached, the test accuracy of the program that performed best on the validation set was recorded. This process was repeated for 150 independent runs, each with a random seed.

Note that in all cases the intial population was created at random, with no control over the amount of overlap between class graphs.

Table 1: Parameter Configurations

Parameter	Basic LGP	Class Graph Crossover	Selective Crossover
Population	500	500	500
Max Gens	200	200	200
Normal Mutation	60%	60%	60%
Elitism	10%	10%	10%
Conventional Crossover	30%	0%	0%
Class Graph Crossover	0%	30%	0%
Selective Crossover	0%	0%	30%
Max Size	32	32	32
Tournament Size	4	4	4
Runs	150	150	150

6. RESULTS AND DISCUSSION

It is the applicability of the LGP approach to multiclass classification and the efficacy of the two novel crossover operators for object recognition tasks that is the subject of this work. The work is firstly placed in a wider context by comparison with TGP approaches, including basic LGP algorithm in section 6.1 and the updated approach in section 6.4. However, it is beyond the scope of the paper to compare all variants of TGP with all variants of LGP across all multiclass problems in order to determine the most applicable approach for any given sub-problem. Section 6.2 presents results for the Class Graph Crossover compared with the basic LGP approach and section 6.3 extends this to compare results for the Selective Crossover.

6.1 Basic LGP vs. TGP

Table 2 shows a comparison between the basic LGP and the basic TGP approach averaged over 150 runs. In the last column "Significant?", "yes" means the results of the two methods are significantly different against the standard T/Z-test with the two-side 95% confidence level, "+" means that the result of the method in the right side is significantly better than that of the left side, and "−" refers to that the right method achieved significantly better result than the left method. In this table, for all the three datasets here, "+yes" suggests that the tree-based GP simply adapted to multiclass problems is outperformed significantly by the basic LGP, see table 2.

However, these results are slightly "misleading" as adjusting TGP by using Probabilistic Multiclass methods (one of the best TGP methods for multiclass classification) shows that (advanced) TGP performs similarly to basic LGP, see table 3. There is no significant difference on one problem domain, (advanced) TGP is significantly better on another domain and basic LGP significantly better on the other do-

Table 2: *Classification Accuracy on the test set:* Basic TGP vs basic LGP

Image Data Set	Basic TGP		Basic LGP		Significant?
	Mean	S.D.	Mean	S.D.	
Artificial Characters	55.91%	8.79%	82.02%	5.72%	+yes
Image Segmentation	68.69%	7.51%	75.46%	2.81%	+yes
Digit Recognition	45.20%	8.34%	65.46%	3.64%	+yes

main. The hypothesised improvements to LGP through the novel tailored crossover operators are now investigated in the rest of this section.

Table 3: *Classification Accuracy on the test set:* TGP with Probability Multiclass (PM) vs basic LGP

Image Data Set	TGP with PM		Basic LGP		Significant?
	Mean	S.D.	Mean	S.D.	
Artificial Characters	81.83%	5.19%	82.02%	5.72%	no
Image Segmentation	85.99%	8.25%	75.46%	2.81%	−yes
Digit Recognition	50.66%	8.38%	65.46%	3.64%	+yes

6.2 LGP with Class Graph Crossover vs. LGP with Canonical Crossover

Table 4 compared the results of LGP with the canonical crossover (basic LGP) and LGP with Class Graph Crossover (CGC) on the three problems on the *test set* averaged over 150 runs. These results show that LGP using CGC demonstrates improved classification accuracy over LGP using canonical crossover on all test data sets, and the improvement is statistically significant in all three problems. These results are an excellent indication that LGP using CGC is superior to LGP using conventional crossover for solving multiclass classification problems.

Table 4: *LGP Classification Accuracy on the test set:* canonical crossover vs. Class Graph Crossover (CGC)

Data Set	LGP		LGP + CGC		Significant?
	Mean	S.D.	Mean	S.D.	
Artificial Characters	82.02%	5.72%	84.61%	5.73%	+yes
Image Segmentation	75.46%	2.81%	76.30%	3.90%	+yes
Digit Recognition	65.46%	3.64%	68.04%	3.86%	+yes

6.3 LGP with Selective Crossover vs. LGP with Class Graph Crossover

The results in table 5 were obtained by running LGP with CGC and LGP with Selective Crossover (SC) on the three problems averaged over 150 runs. These results show that LGP using (SC) demonstrates improved classification accuracy over LGP using CGC on all test data sets, and the improvement is statistically significant in all three problems. These results are an excellent indication that LGP using SC is superior to LGP using CGC for solving multiclass object classification problems.

Table 5: *LGP Classification Accuracy on the test set:* Class Graph Crossover (CGC) vs Selected Crossover (SC)

Data Set	LGP + CGC		LGP + SC		Significant?
	Mean	S.D.	Mean	S.D.	
Artificial Characters	84.61%	5.73%	86.65%	4.74%	+yes
Image Segmentation	76.30%	3.90%	77.47%	3.91%	+yes
Digit Recognition	68.04%	3.86%	69.16%	3.21%	+yes

6.4 Further Discussion

It has been shown that the novel crossover operators improve the multiclass performance of LGP on the test domains. The improvements are now compared with one of the best TGP method (with Probabililty Multiclass), see section 6.1, to place them in context. Table 6 shows that now the LGP based technique is significantly better than TGP for the Artificial Characters dataset, whereas previously it was equivalent. However, the advanced TGP method is still significantly better for the Image Segmentation dataset despite the improvements. This might be because the Image Segmentation problem has a relatively small number of classes where TGP with PM can usually perform well, but this needs to be further investigated in the future. Nevertheless, although the novel operators have improved the performance of LGP, there are further performance improvements possible. Also the TGP approach is not dominated by LGP for all problem types.

Table 6: *Classification Accuracy on the test set:* Best TGP results vs LGP with Advanced Crossover (SC)

Data Set	TGP + PM		LGP + SC		Significant?
	Mean	S.D.	Mean	S.D.	
Artificial Characters	81.83%	5.19%	86.65%	4.74%	+yes
Image Segmentation	85.99%	8.25%	77.47%	3.91%	−yes
Digit Recognition	50.66%	8.38%	69.16%	3.21%	+yes

7. CONCLUSIONS AND FUTURE WORK

Multiclass classification problems occur naturally in many computer vision applications with the automatic generation of good solutions being of great importance. Currently, GP methods are not favored for solving multiclass classification problems due in large part to the program structure of the conventional TGP method. Experiments with an alternative form of GP, LGP, have shown a plausibly better program structure and learning algorithm for these types of problems.

The conventional LGP crossover operator exchanges program instructions at random between two LGP programs. This is problematic because exchanging instructions at random is likely to disrupt good building blocks of code. Hence a new form of crossover was introduced, called Class Graph Crossover (CGC), where code exchanges only occur between two code sequences that determine the same feature. The results show that LGP with CGC has significantly superior performance to conventional LGP on all problems tested.

Current advances in crossover focus on improving TGP performance by increasing the likelihood that the offspring

produced by crossover have high fitness. The issue with these methods is that they dramatically increase the computational cost of the crossover operator because they calculate the fitness of many offspring. A further developed crossover operator, called Selective Crossover, addresses this problem by using a computationally cheap heuristic to predict offspring fitness. The results show that selective crossover outperforms both conventional LGP crossover and Class Graph Crossover by a significant amount on all problems tested.

It is noted that an advanced TGP algorithm outperforms the advanced LGP algorithms developed here in the case of the dataset with the lowest number of classes. It is considered that bespoke TGP instances will be complementary to tuned LGP algorithms for multiclass problems.

There are many rich areas of future work which follow naturally from this project. It would be interesting to analyze the overlap between class graphs during evolution. It would also be worthwhile to empirically demonstrate the efficiency of the selective crossover operator. More work could be done to determine to what extent the claim that class graph crossover aids in repairing building blocks holds. Finally it would be interesting to compare how all of these factors are impacted by the number of classes.

8. REFERENCES

[1] 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 Publishers. 1998.

[2] M. Brameier and W. Banzhaf. *Linear Genetic Programming*. Number XVI in Genetic and Evolutionary Computation. Springer, 2007.

[3] H. J. Cordell. Epistasis: What it means, what it doesn't mean, and statistical methods to detect it in humans. *Human Molecular Genetics*, 11(10):2463–2468, 2002.

[4] C. Fogelberg and M. Zhang. Linear genetic programming for multi-class object classification. In S. Zhang and R. Jarvis, editors, *Proceedings of 18th Australian Joint Conference on Artificial Intelligence,* , pages 369–379, 2005.

[5] F. D. Francone, M. Conrads, W. Banzhaf, and P. Nordin. Homologous crossover in genetic programming. In W. Banzhaf, et al. editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1021–1026, 1999.

[6] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, 1975.

[7] B. Hutt and K. Warwick. Synapsing variable-length crossover: Meaningful crossover for variable-length genomes. *IEEE Trans. Evolutionary Computation*, 11(1):118–131, 2007.

[8] W. Kantschik and W. Banzhaf. Linear-graph GP—A new GP structure. In J. A. Foster, et al. editors, *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, pages 83–92, 2002.

[9] J. R. Koza. *Genetic programming : on the programming of computers by means of natural selection*. MIT Press, 1992.

[10] K. Krawiec and B. Bhanu. Visual learning by evolutionary and coevolutionary feature synthesis. *IEEE Transactions on Evolutionary Computation*, 11(5):635–650, 2007.

[11] W. B. Langdon. Size fair and homologous tree genetic programming crossovers. *Genetic Programming and Evolvable Machines*, 1(1/2):95–119, 2000.

[12] T. Loveard. *Genetic Programming for Classification Learning Problems*. PhD thesis, RMIT University, School of Computer Science and Information Technology, 2003.

[13] T. Loveard and V. Ciesielski. Representing classification problems in genetic programming. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1070–1077, 2001.

[14] P. Nordin. A compiling genetic programming system that directly manipulates the machine code. In K. E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 14, pages 311–331. MIT Press, 1994.

[15] G. Olague Caballero, E. Romero, L. Trujillo, and B. Bhanu. Multiclass object recognition based on texture linear genetic programming. In M. Giacobini, et al. editors, *Applications of Evolutionary Computing, EvoWorkshops2007*, pages 291–300, 2007.

[16] G. Olaguea, S. Cagnoni, and E. Lutton. (eds.) special issue on evolutionary computer vision and image understanding, pattern recognition letters.**27**(11), 2006.

[17] R. Poli and N. F. McPhee. General schema theory for genetic programming with subtree-swapping crossover: Part I. *Evolutionary Computation*, 11(1):53–66, Mar. 2003.

[18] M. E. Roberts and E. Claridge. A multistage approach to cooperatively coevolving feature construction and object detection. In F. Rothlauf, et al. editors, *Applications of Evolutionary Computing, EvoWorkshops2005* pages 396–406, 2005.

[19] C. B. S. Hettich and C. Merz. UCI repository of machine learning databases, 1998.

[20] A. Song. *Texture Classification: A Genetic Programming Approach*. PhD thesis, Department of Computer Science, RMIT University, Melbourne, Australia, 2003.

[21] W. A. Tackett. Genetic programming for feature discovery and image discrimination. In S. Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 303–309, 1993.

[22] W. A. Tackett. *Recombination, Selection, and the Genetic Construction of Computer Programs*. PhD thesis, University of Southern California, Department of Electrical Engineering Systems, USA, 1994.

[23] W. A. Tackett and A. Carmi. The unique implications of brood selection for genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, 1994.

[24] P. A. Whigham. Grammatically-based genetic programming. In J. P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, 1995.

[25] M. Zhang and V. Ciesielski. Genetic programming for multiple class object detection. In N. Foo, editor, *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence*, pages 180–192, 1999.

[26] M. Zhang, V. B. Ciesielski, and P. Andreae. A domain-independent window approach to multiclass object detection using genetic programming. *EURASIP Journal on Applied Signal Processing*, 2003(8):841–859, 2003.

[27] M. Zhang, X. Gao, and W. Lou. A new crossover operator in genetic programming for object classification. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 37(5):1332–1343, 2007.

[28] M. Zhang and W. Smart. Multiclass object classification using genetic programming. In G. R. Raidl, et al. editors, *Applications of Evolutionary Computing, EvoWorkshops2004*, pages 369–378, 2004.

[29] M. Zhang and W. Smart. Using gaussian distribution to construct fitness functions in genetic programming for multiclass object classification. *Pattern Recognition Letters*, 27(11):1266–1274, 2006.