Complex 2004

Proceedings of the 7th Asia-Pacific
Conference on Complex Systems
Cairns Convention Centre, Cairns, Australia
6-10th December 2004

Continuously Evolving Programs in Genetic Programming Using Gradient Descent

Will Smart[1] and Mengjie Zhang[1]
[1]*School of Mathematics, Statistics and Compute Science*
Victoria University of Wellington,
P. O. Box 600, Wellington, New Zealand
`Email: {smartwill,mengjie}@mcs.vuw.ac.nz`

Abstract

This paper describes an approach to the use of gradient descent search in genetic programming for continuously evolving genetic programs for object classification problems. An inclusion factor is introduced to each node in a genetic program and gradient descent search is applied to the inclusion factors. Three new on-zero operators and two new continuous genetic operators are developed for evolution. This approach is examined and compared with a basic GP approach on three object classification problems of varying difficulty. The results suggest that the new approach can evolve genetic programs continuously. The new method which uses the standard genetic operators and gradient descent applied to the inclusion factors substantially outperforms the basic GP approach which uses the standard genetic operators but does not use the gradient descent and inclusion factors. However, the new method with the continuous operators and the gradient descent on inclusion factors decreases the performance on all the problems.

1. Introduction

Since the early 1990s, there has been a number of reports on applying genetic programming (GP) techniques to object recognition problems (Andre, 1994; Howard et al., 1999; Koza, 1992; Loveard and Ciesielski, 2001; Song et al., 2002; Tackett, 1993; Winkeler and Manjunath, 1997; Zhang and Ciesielski, 1999). Typically, these GP systems used either high level or low level image features as the terminal set, arithmetic and conditional operators as the function set, and classification accuracy, error rate or similar measures as the fitness function. During the evolutionary process, selection, crossover and mutation operators were applied to the genetic beam search to find good solutions.

While the GP approach has achieved some success in a number of application areas, the performance of the programs evolved in adjacent generations is often unstable. This is mainly because the GP evolutionary beam search process improves the programs in a discontinuous way. During the evolutionary process, they did not use the existing heuristics inside individual programs, for example, the gap between the actual outputs of the programs and the target outputs.

Gradient descent is a long term established search/learning technique and commonly used to train multilayer feed forward neural networks (Rumelhart et al., 1986). This algorithm can guarantee to find a local minima for a particular task. While the local minima is not the best solution, it often meets the request of that task. A main characteristic of gradient descent search is that the solutions can be improved gradually and steadily in a continuous way.

Gradient descent search has been applied to numeric terminals (Zhang and Smart, 2004) or constants (Ryan and Keijzer, 2003) of genetic programs in GP. In these approaches, gradient descent search is locally applied to individual programs in a particular generation and the constants in a program are updated in a continuous manner. However, the programs are still globally updated by genetic beam search in a discontinuous manner.

1.1 Goals

The goal of this paper is to investigate a continuous approach to the use of gradient descent search for evolving genetic programs in GP. To apply gradient descent to genetic programs, a new parameter, *inclusion factor*, is introduced to each node so that a partial program tree can be changed according to the gradient of the inclusion factors. To avoid discontinuous updating of programs in the evolutionary process, a set of new genetic operators are developed. This approach will be examined on three object classification problems and compared with the standard GP approach (referred to as *the basic GP approach*). Specifically, we are interested in:

- How the inclusion factor can be introduced to a GP node and how this factor can be updated using gradient descent.
- How the new genetic operators can be developed so that the program performance can be improved based only on gradient descent search on the inclusion factors.
- Whether the genetic programs can be evolved and improved continuously.
- Whether the GP approach with the inclusion factor to which gradient descent search is applied outperforms the basic GP approach.
- Whether the GP approach with both the inclusion factor and the new operators outperforms the basic GP approach.

1.2 Structure

The remainder of the paper is organised as follows. Section 2 presents a brief overview of the basic GP approach. Section 3 describes the introduction of the inclusion factor and the gradient descent algorithm applied to the inclusion factor. Section 4 describes the new genetic operators. Section 5 describes object classification data sets on which the approach is examined. Section 6 presents the experiment configuration and results. Section 7 draws conclusions and describes future work directions.

2. GP Applied to Object Classification

In the basic GP approach, we used the tree-structure to represent genetic programs (Koza, 1992). The ramped half-and-half method was used for generating programs in the initial population and for the mutation operator (Banzhaf et al., 1998). The proportional selection mechanism and the reproduction, crossover and mutation operators (Koza, 1994) were used in the learning and evolutionary process.

2.1 Primitive Sets

For object classification problems, terminals generally correspond to image features. In this approach, we used pixel level, domain independent statistical features (referred to as *pixel statistics*) as terminals and we expect the GP evolutionary process can automatically select features that are relevant to a particular domain to construct good genetic programs.

Four pixel statistics are used in this approach: the average intensity of the whole object cutout image, the variance of intensity of the whole object cutout image, the average intensity of the central local region, and the variance of intensity of the central local region. Since the range of these four features are quite different, we linearly normalised these feature values into the range [-1, 1] based on all object image examples to be classified.

In addition, we also used some constants as terminals. These constants are randomly generated using a uniform distribution. To be consistent with the feature terminals, we also set the range of the constants as [-1, 1].

The function set uses two standard arithmetic operators to be consistent with the new approach (see table 1 in section 3.1): addition and multiplication. They have the usual meanings of addition and multiplication with two arguments.

2.2 Fitness Function

We used classification accuracy on the training set as the fitness function. The classification accuracy of a genetic program classifier refers to the number of object images that are correctly classified by the genetic program classifier as a proportion of the total number of object images in the training set.

In this approach, we used a variant version of the *program classification map* (Zhang et al., 2003) to perform object classification. This variation situates class regions sequentially on the floating point number line. The object image will be classified to the class of the region that the program output with the object image input falls into. Class region boundaries start at some negative number, and end at the same positive number. Boundaries between the starting point and the end point are allocated with an identical interval of 1.0. For example, a five class problem would have the classification map shown in figure 1.

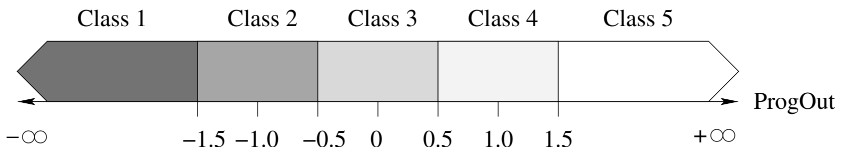

Figure 1. *A variation of the program classification map.*

The proposed new approach used the same primitive sets and the fitness function as the basic GP approach. However, the new approach introduced an inclusion factor into the program nodes, applied gradient descent search to the inclusion factors, and introduced new continuous operators. This will be described in section 3 and section 4.

3. Gradient Descent Applied to Program Nodes

In order to make genetic programs improve continuously, gradient-descent search is chosen to make partial changes of the certain parts of genetic programs during evolution. To do this, we introduced an *inclusion factor* to the nodes of the genetic programs. This section describes the properties of inclusion factors and how gradient descent is applied to the inclusion factors.

3.1 Inclusion Factors

This approach introduced an inclusion factor to each node except the root in a program tree. The inclusion factor of a node is defined as a floating point variable ranged from 0 to 1 and determines the *proportion* of the node really included in the program. An inclusion factor with a value of 1 for a node means that the node will be evaluated as normal in the standard genetic program, while a value 0 means that the node will not be included at all when the node is evaluated. We expect that with the inclusion factor, certain parts of the whole program can be updated in a continuous way.

Due to the introduction of inclusion factor, the functions used in the program need to be redefined. Table 1 shows the return values of the functions with and without the inclusion factor. In the table, a_n is the evaluated value of the n'th argument and x_n is the inclusion factor of the n'th argument's node.

Table 1. *Primitive functions with and without inclusion factors.*

Primitive functions	**Function without inclusion factors**	**Function with inclusion factors**
Addition	$a_1 + a_2$	$x_1a_1 + x_2a_2$
Multiplication	a_1a_2	$(1 + x_1(a_1 - 1))(1 + x_2(a_2 - 1))$

Note that only the multiplication and addition functions are used here. The major consideration is that they are relatively easy to implement and they can meet the requirements of many relatively uncomplex application problems. While they might not be sufficient for some difficult problems, this is beyond the scope of this paper. This paper will investigate the idea first and if the idea works successfully, we will investigate more functions in the future.

3.2 Gradient-Descent of Inclusion Factors

This section describes how to apply gradient-descent search to inclusion factors to improve the performance of a genetic program.

In this approach, gradient-descent is applied to changing the values of the inclusion factors. It is assumed that a continuous cost surface C can be found to describe the performance of a program at a particular classification task for all possible values for the inclusion factors. To improve the system performance, the gradient descent search is applied to taking steps “downhill” on the C from the current inclusion factor.

The gradient of C is found as the vector of partial derivatives with respect to the parameter values. This gradient vector points along the surface, in the direction of maximum-slope at the point used in the derivation. Changing the parameters proportionally to this vector (negatively, as it points to “uphill”) will move the system down the surface C. If we use x_i to represent the value of the ith inclusion factor and y to represent the output of the genetic program P, then the distance moved (the change of x_i) should therefore be:

$$\Delta x_i = -\alpha \cdot \frac{\partial C}{\partial x_i} = -\alpha \cdot \frac{\partial C}{\partial y} \cdot \frac{\partial y}{\partial x_i} \tag{1}$$

where α is a search factor. In the rest of this section, we will address the three parts — $\frac{\partial C}{\partial y}$, $\frac{\partial y}{\partial x_i}$ and α.

3.2.1 Cost Surface

We used the sum-squared error as the cost surface C, as shown in equation 2.

$$C = \frac{\sum_{j=1}^{N} (y_j - Y_j)^2}{2} \tag{2}$$

where Y_j is the desired program output for training example j, y_j is the actual calculated program output for training example j, and N is the number of training examples.

Accordingly, the partial derivative of the cost function with respect to the genetic program for training example j would be:

$$\frac{\partial C}{\partial y_j} = \frac{\partial(\frac{(y_j - Y_j)^2}{2})}{\partial y_j} = y_j - Y_j \tag{3}$$

The corresponding desired output Y is calculated as follows:

$$Y_j = \textbf{class} - \frac{\textbf{numclass} + 1}{2} \tag{4}$$

where **class** is the class label of the object and **numclass** is the total number of classes. For example, for a five class problem as described in section 1, the desired outputs are $-2, -1, 0, 1$ and 2 for object classes 1, 2, 3, 4 and 5, respectively.

3.2.2 Partial Derivative $\frac{\partial y_j}{\partial x_i}$

For presentation convenience, y_j will be written as y with the pattern j omitted.

In order to illustrate the calculation of the derivative of the program output by a change in the value of an inclusion factor, the program shown in figure 2 is used as an example.

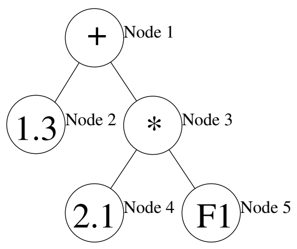

Figure 2. *An example program.*

If we use O_i to represent the output of node i, then the partial derivatives of the genetic program with respect to the inclusion factor x_i for node i will be (we use node 5 as an example):

$$\begin{aligned} \frac{\partial y}{\partial x_5} &= \frac{\partial O_1}{\partial x_5} = \frac{\partial O_1}{\partial O_3} \cdot \frac{\partial O_3}{\partial x_5} \\ &= \frac{\partial(x_2 O_2 + x_3 O_3)}{\partial O_3} \cdot \frac{\partial(1 + x_4(O_4 - 1))(1 + x_5(O_5 - 1))}{\partial x_5} \\ &= x_3 \cdot (1 + x_4(O_4 - 1))(O_5 - 1) \end{aligned}$$

Since x_3 and x_4 are known and O_4 and O_5 can be obtained during evaluation of the program, the formula is readily calculated.

In this way the appropriate derivative for any inclusion factor, in a program of any depth, can be found using the chain rule and some simple derived operators.

3.2.3 Search Factor α

The search factor α in equation 1 was defined to be proportional to the inversed sum of the square gradients on all inclusion factors along the cost surface, as shown in equation 5.

$$\alpha = \eta \div \sum_{i=1}^{M} (\frac{\partial C}{\partial x_i})^2 \tag{5}$$

where M is the number of inclusion factors in the program, and η is a learning rate defined by the user. The intuition behind this is that at learning rate 1.0, if all the inclusion factors are independent, the new output of the genetic program with the change by this gradient descent algorithm would be very close to (ideally the same as) the desired target output.

3.2.4 Summary of the Gradient Descent Algorithm

- Evaluate the program, save the outputs of all nodes in the program.
- Calculate the partial derivative of the cost function on the program $\frac{\partial C}{\partial y}$ using equations 3 and 4.
- Calculate the partial derivatives of the program on inclusion factors $\frac{\partial y}{\partial x_i}$ using the chain rule and table 1.
- Calculate the search factor α using equation 5.
- Calculate the change of each inclusion factor using equation 1.
- Update the inclusion factors using $(x_i)_{new} = x_i + \Delta x_i$.

The application of gradient descent on inclusion factors allows for the entire program space to be searched in a continuous way.

In order to stabilise the gradient-descent search and to reduce the computation cost, not all inclusion factors are changed on each iteration. Currently the three greatest and three least components in the gradient vector are used, so a maximum of six parameters are changed in an iteration.

4. New Genetic Operations for Continuous Evolution

The use of inclusion factor has an important property. When a node has an inclusion factor of 0, any changes may be made to the node, or its arguments (if it is a function), while no change will be noticed in the program output. This allows the structure of the program to be changed in certain ways, while not changing output discontinuously as would normally occur in the standard GP evolutionary process.

We developed two kinds of operators for continuous variation of programs using this property of the inclusion factor: *on-zero operators* and new *genetic operators*.

4.1 On-Zero Operators

When an inclusion factor reaches zero (or negative and force to zero) through gradient descent, one of the following operators is selected and applied:

- **Deletion:** The parent of the node with an inclusion factor of zero is replaced by its other child, as shown in figure 3(a). Note that this is only applied to parent nodes with two arguments, one of which has an inclusion factor of zero and the other of one.
- **On-zero-mutation:** The node with an inclusion factor of zero is replaced by a randomly generated subtree. This operation is shown in figure 3(b).
- **On-zero-crossover:** The node with an inclusion factor of zero is replaced by a randomly selected subtree from the population. This operation is shown in figure 3(c).

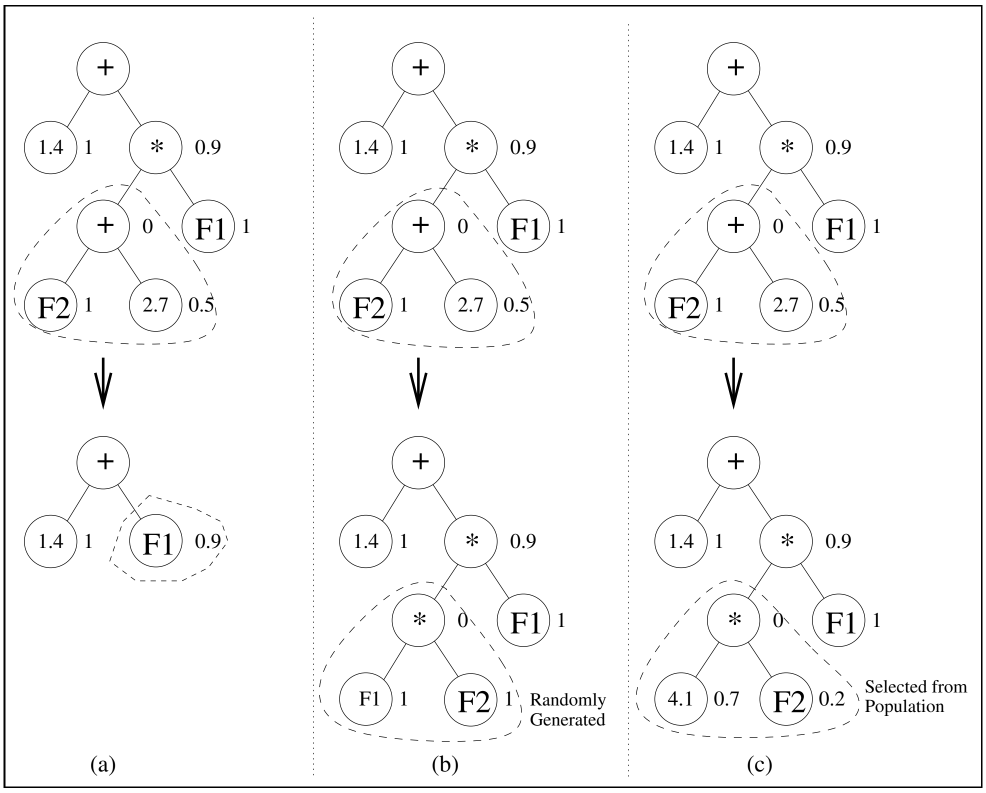

Figure 3. *Examples using the "on-zero" operators.*

4.2 Continuous Genetic Operators

The operators included in this section are applied to programs in the same circumstance as the normal genetic operators are in standard GP. Each generation, some proportion of the population are produced through reproduction (which is the same as in standard GP), some proportion through the new continuous mutation, and some proportion through the new continuous crossover.

The new continuous mutation operator is applied to a randomly selected node in a selected program. A mutation point is randomly chosen and the selected subtree in the program is replaced with a randomly selected function. The original sub-tree at the mutation point then forms the right child (one argument) of the function with an inclusion factor of 1. A randomly generated subtree with an inclusion factor of 0 for the subtree root and 1 for all nodes deeper in the subtree forms the left child (the other argument) of the function. This process is shown in figure 4(a).

In the new continuous crossover operator, a crossover points is randomly chosen from a selected program in the tournament. Similarly to the new mutation operator, a crossover point is randomly chosen and the selected subtree in the program is replaced with a random function and the original subtree at the crossover point then forms the right child (one argument) with an inclusion factor of 1. Unlike the new mutation operator, a randomly selected subtree in the tournament with an inclusion factor of zero for the subtree root forms the (left) child of the function. This process is shown in figure 4(b). The new crossover operator is quite different from the standard one in that the new operation only produces one offspring.

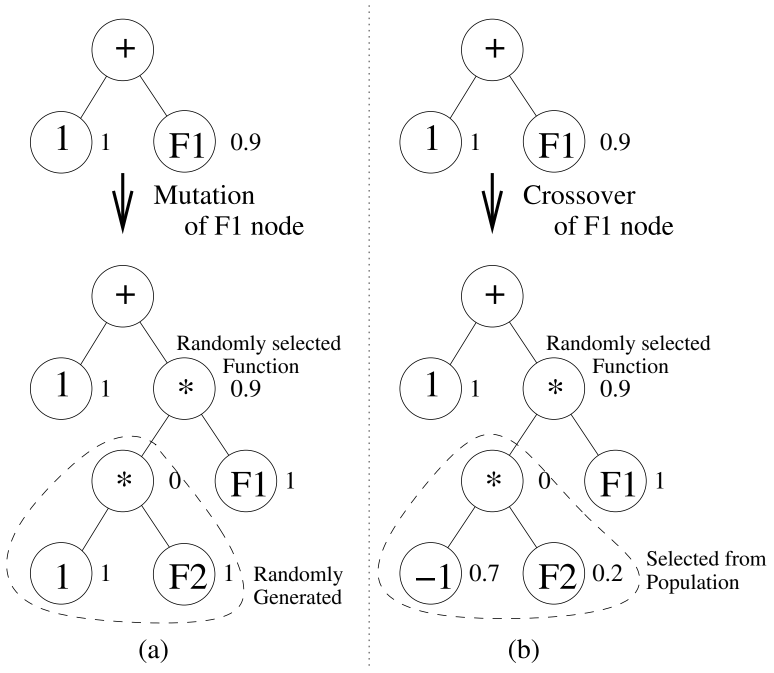

Figure 4. *Examples using the new continuous genetic operators.*

It is important to note that these new operators are different in nature from the standard genetic operators in that they do not change the return value of the programs (output) and the original output values remain. In this way, the programs are updated continuously according to the gradient descent algorithm described in the last section. However, the program structures were changed by these operators and we expect that the advantages of the conventional genetic operators can stay during evolution and the structure can be updated in a continuous way.

5. Data Sets

We used three data sets providing object classification problems of varying difficulty in the experiments. Example images are shown in figure 5.

5.1 Computer Generated Shape Data Set

The first set of images (figure 5a) 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. Three classes of 960 small objects were cut out from those images to form the classification data set. The three classes are: black circles, grey squares, and light circles. For presentation convenience, this dataset is referred to as *shape*.

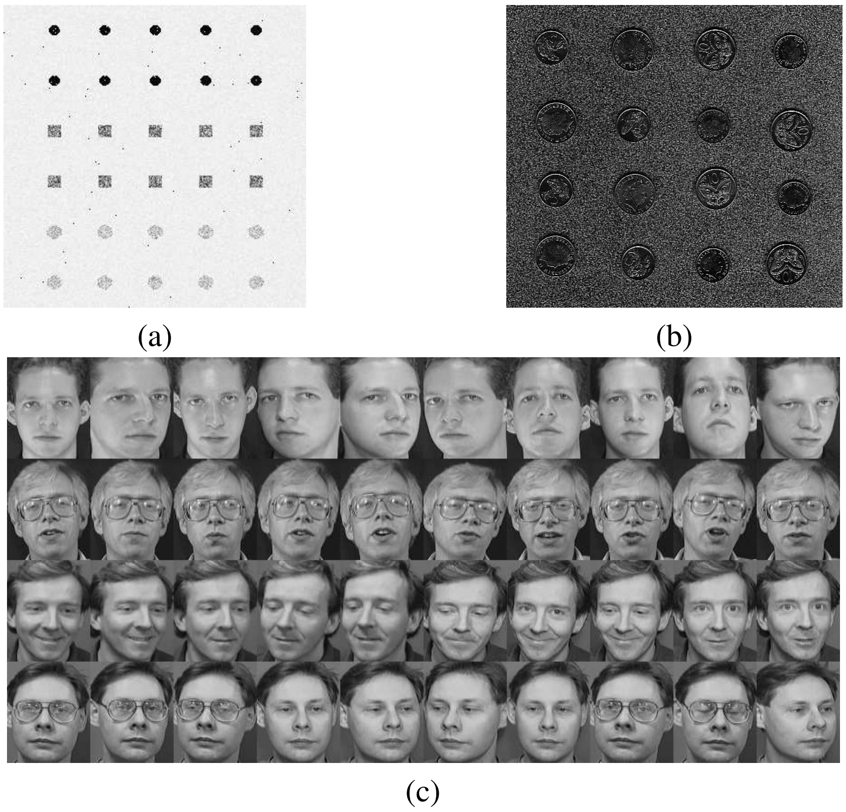

Figure 5. *Sample image data sets. (a) Shape; (b) Coin; (c) Face.*

5.2 NZ Coin Data Set

The second set of images (figure 5b) contains scanned 5 cent and 10 cent New Zealand coins. The coins were located in different places with different orientations and appeared in different sides (head and tail). In addition, the background was cluttered. We need to distinguish different coins with different sides from the background. Five classes of 801 object cutouts were created: 160 5-cent heads, 160 5-cent tails, 160 10-cent heads, 160 10-cent tails, and the cluttered background (161 cutouts). Compared with the *shape* data set, the classification problem in this data set is much harder. Although these are still regular, man-made objects, the problem is very hard due to the noisy background and the low resolution.

5.3 Human Face Data Set

The third data set consists of 40 human faces (figure 5c) taken at different times, varying lighting slightly, with different expressions (open/closed eyes, smiling/non-smiling) and facial details (glasses/no-glasses). These images were collected from the first four directories of the ORL face database (Samaria and Harter, 1994). All the images were taken against a dark homogeneous background with limited orientations. The task here is to distinguish those faces into the four different people.

For the shape and the coin data sets, the objects were equally split into three separate data sets: one third for the training set used directly for learning the genetic program classifiers, one third for the validation set for controlling overfitting, and one third for the test set for measuring the performance of the learned program classifiers. For the faces data set, due to the small number of images, ten-fold cross validation was applied.

6. Experimental Results and Discussion

6.1 Experiment Configuration

The parameter values used in this approach are shown in table 2. The evolutionary process is run for a fixed number (*max-generations*) of generations, unless it finds a program that solves the classification perfectly (100% accuracy), at which point the evolution is terminated early.

Each generation, the most-fit individual in the population was used to get validation set and test set accuracies. In results for the shape and coin data sets, the test accuracy given is the *test accuracy at best validation*, i.e. the test set accuracy in the generation that had the best validation set accuracy. Due to the use of a validation set, no overfitting should occur. Overfitting in a generation will cause bad accuracy on the validation set, so the generation's test accuracy will not be used for the results of the run.

In the face data set, due to the lack of a validation set, the results given are simply the best test set accuracy found in any generation.

The results given are those at convergence, so long as convergence occurs before the number of generations exceeds *max-generations*. An indication as to whether convergence is generally reached by evolutions may be found by comparing the *generations* statistic in results to the *max-generations* value. If they are close, then most runs did not get 100% accuracy on the training set. However, if the *generations* statistic is far below the *max-generations* parameter, then most runs did get 100% accuracy on the training set and did converge.

Table 2. *Parameters used for GP training for the three datasets.*

Parameter Names	Shape	coin	face	Parameter Names	Shape	coin	face
population-size	300	500	500	reproduction-rate	10%	10%	10%
initial-max-depth	3	4	4	crossover-rate	60%	60%	60%
max-depth	5	7	7	mutation-rate	30%	30%	30%
max-generations	100	100	100	cross-term	15%	15%	15%
object-size	16×16	70×70	92×112	learning rate η	1.0	1.0	1.0

The rest of this section presents a series of results of the new approach on the three object classification data sets. These results are compared with those for the basic GP approach. For all experiments, we run 50 times with random seeds and the average results on the test set were presented.

6.2 Overall Results

For presentation convenience, we used the following terms for the three GP approaches:

- **GP-Standard:** the basic GP approach with the standard genetic operators. No gradient descent is used in this approach.
- **GP-Inclusion:** inclusion factors are introduced to the GP system and the gradient descent search is applied to the inclusion factors. The standard genetic operators are used in the evolutionary process. In this approach, the values of the genetic programs are updated in a continuous way inside a particular generation, but in a discontinuous way across the whole evolutionary process.

- **GP-Continuous:** in this approach, gradient-descent is applied to the inclusion factors and the new/continuous genetic operators are used in the evolutionary process. The return values of the genetic programs are updated in entirely continuous way.

Table 3 shows the overall results of the three GP approaches. The first row shows that, the standard GP system achieved 98.87% accuracy on average on the test set of the shape data set, the average evolutionary training time was 0.24 seconds and average number of generations taken in the evolutionary training process was 23.02.

Table 3. *Comparison of results of the GP approaches on the three data sets.*

Dataset	GP used	Generations	Time (s)	Test Accuracy (%)
Shape	GP-Standard	23.02	0.24	98.87
	GP-Inclusion	11.18	0.33	99.49
	GP-Continuous	61.64	3.94	97.23
Coin	GP-Standard	55.20	0.35	66.24
	GP-Inclusion	62.72	1.49	72.08
	GP-Continuous	56.58	2.12	59.60
Face	GP-Standard	5.94	0.02	79.90
	GP-Inclusion	5.74	0.04	81.60
	GP-Continuous	6.49	0.06	73.40

For the shape data set, the three GP approaches achieved 98.87%, 99.49% and 97.23% accuracy, respectively, showing that the GP approach with gradient descent on inclusion factors and the standard genetic operators (GP-inclusion) performed the best and the new continuous GP approach with gradient descent on inclusion factors and the new continuous operators (GP-continuous) performed the worst. The coin and the face data sets show a similar pattern to the shape data set.

The results suggest that the introduction of inclusion factors to the program nodes and applying gradient descent search to the inclusion factors in genetic programs improves the performance over the standard GP approach, which is consistent with our hypothesis. This is mainly because this new approach takes the advantages of both the genetic beam search and the gradient descent search. While the genetic beam search is still applied to the genetic programs across the whole evolutionary process globally, the gradient descent search is applied to the individual programs locally inside a particular generation.

The results also suggest that the replacement of the standard genetic operators with the new continuous operators weakens the system performance, which is different from our original hypothesis. While this new method did evolve programs with better fitness gradually and continuously, it also lost the advantage of the genetic beam search. In other words, this method in theory only used gradient descent search and degraded the genetic beam search. In addition, only gradient descent search in GP did not do as good a job as the genetic beam search only in GP.

The results also suggest that the two new GP approaches need more time to evolve good genetic programs. In particular, the new continuous GP approach appeared to take much longer to train the programs. Due to the introduction of the inclusion factors to the genetic programs and gradient descent search to the inclusion factors, the new approach *GP-inclusion* spent more time on a single generation than the standard GP. Due to the introduction of the new on-zero and new genetic operators, the new approach *GP-continuous* took even longer time to evolve good programs in the training process.

6.3 Results on Mutation Operator

To examine the effect of the new continuous mutation operator, we used a reproduction rate of 10% and all the rest 90% of programs for mutation. In other words, we did not use any crossover or other operators in the experiment.

Table 4 shows a comparison of the results on the new continuous mutation operator to the standard mutation operator. It does this by sliding the proportion of mutations done using the new method from none to all.

Table 4. *Effect of the new continuous mutation operator.*

Dataset	Rates for new mutation	Generations	Time (s)	Test Accuracy (%)
Shape	0%	15.36	0.67	99.49
	25%	13.82	0.59	99.43
	50%	19.94	0.91	99.32
	75%	19.68	0.92	99.16
	100%	57.60	4.76	97.39
Coin	0%	58.84	1.75	69.79
	25%	63.10	2.16	70.41
	50%	63.86	2.22	68.60
	75%	67.22	2.59	67.32
	100%	61.10	2.67	59.44
Face	0%	6.55	0.06	80.50
	25%	7.17	0.06	80.10
	50%	7.82	0.07	80.55
	75%	10.51	0.10	80.20
	100%	4.75	0.04	72.35

As can be seen from table 4, if all mutations (100%) are done using the new continuous operator, the final performance of the system will be reduced. The reduction of performance is quite substantial in the relatively difficult tasks.

An increase of the rate on the new mutation operator against the standard mutation operator generally also results in an increase of training time in the evolutionary process.

These results suggest that the new operator has very little positive effect on performance in terms of the system results. Although it does introduce new genetic materials and give support to allow the return values of genetic programs updated in a continuous way, it reduces the system performance overall.

6.4 Results on Crossover Operator

To examine the effect of the new continuous crossover operator, we used a reproduction rate of 10% and all the rest 90% of programs for crossover. In other words, we did not use any mutation or other operators in the experiment.

Table 5 shows a comparison of the results on the crossover operator with the standard crossover operator. It does this by sliding the proportion of crossovers done using the new crossover operator from none to all.

Similarly to the new mutation operator, if all crossovers (100%) are done using the new operator, the final performance of the system will be reduced. The reduction of performance is

Table 5. *Effect of the new continuous crossover operator.*

Dataset	Rate for new Crossover	Generations	Time (s)	Test Accuracy (%)
Shape	0%	13.00	0.26	99.41
	25%	18.20	0.45	99.16
	50%	26.26	0.73	98.70
	75%	28.50	0.98	98.67
	100%	50.60	2.61	96.89
Coin	0%	33.42	0.38	61.38
	25%	32.04	0.40	59.15
	50%	45.24	0.75	61.40
	75%	40.98	0.73	59.96
	100%	44.84	1.14	55.66
Face	0%	5.07	0.02	80.65
	25%	5.29	0.02	81.15
	50%	7.44	0.04	81.20
	75%	7.21	0.04	80.05
	100%	3.62	0.02	72.05

quite substantial in the relatively difficult problems.

An increase of the rate on the new crossover operator against the standard crossover operator generally also results in an increase of training time in the evolutionary process.

For the shape data set, as the proportion of crossovers done using the new operator increases, the performance will be lowered. For the two harder data sets, as long as some crossovers are done using the standard operator, the performance is very similar. In particular, in the two harder data sets, the new crossover operators did play certain positive role: if half of the operators were done with the new crossover operator, the performance could be improved, but the improvement is very little. This suggests that the new continuous crossover operator can introduce new genetic materials, but this positive effect is very much offset by the redundancy introduced by the operator.

6.5 Results for On-Zero Operators

To examine the effect of the on-zero operator, we used the following parameter settings: a reproduction rate of 10%, a rate of 60% on the new crossover operator, and a rate of 30% on the new mutation operator.

Table 6 shows a comparison of the on-zero operators, deletion, on-zero-crossover and on-zero-mutation, on the three data sets.

As can be seen from this table, it seems that these operators has little effect on the accuracy of the system overall. The most clear trend is that the deletion operator has a small but positive effect: the system with the deletion operator generally resulted in better performance than without. This is mainly because the deletion operator simplified the genetic programs by removing the redundancy of certain branch of a program after the inclusion factor of that part reaches zero. While the on-zero-crossover and on-zero-mutation operators introduced new genetic materials to a program, this effect was largely offset by the redundancy.

Table 6. *Effect of the on-zero operators.*

Dataset	On-zero operators			Generations	Time (s)	Test Accuracy (%)
	Delete	Mutation	Crossover			
Shape	off	off	on	58.68	8.09	96.86
		on	off	61.18	5.89	96.26
			on	58.22	7.09	96.97
	on	off	off	61.64	3.94	97.23
			on	60.16	6.35	96.14
		on	off	56.78	3.67	95.35
			on	59.02	6.32	97.21
Coin	off	off	on	39.18	3.28	54.31
		on	off	50.12	2.84	55.54
			on	41.08	3.18	55.98
	on	off	off	55.38	1.81	58.25
			on	51.12	3.06	56.01
		on	off	56.58	2.12	59.60
			on	50.72	3.06	56.34
Face	off	off	on	4.44	0.07	72.30
		on	off	3.71	0.04	72.20
			on	5.29	0.08	72.60
	on	off	off	5.83	0.05	73.30
			on	6.87	0.09	73.00
		on	off	6.49	0.06	73.40
			on	5.65	0.08	72.80

7. Conclusions

The goal of this paper was to investigate a continuous approach to the use of gradient descent search for evolving genetic programs in GP. The goal was achieved by introducing an inclusion factor to the program nodes, applying gradient descent search to the inclusion factor, and developing and applying new continuous genetic and on-zero operators to the programs across the evolutionary process.

During the development of this approach, two new methods, standard GP operators with the gradient descent search applied to the inclusion factors (*GP-inclusion*) and the new continuous GP operators with gradient descent search on the inclusion factor (*GP-continuous*), were examined and compared with the standard GP approach on three object classification problems of varying difficulty. The GP-inclusion method, which applied the genetic beam search globally across the whole evolutionary process and applied the gradient descent search locally to the individual programs inside a particular generation, greatly outperformed the the standard GP approach. However, the GP-continuous method decreased the performance on all data sets because it removed advantages of the genetic beam search from the evolution. The results also suggest that GP with the gradient descent search only cannot perform as well as the GP with genetic beam search only.

The majority of the new continuous operators developed here had very little positive effect due to adding new redundancy to the programs. The deletion operator, on the other hand, did play a positive role during evolution since it removes the unnecessary redundancy in the programs when the inclusion factor of a certain part becomes zero.

For future work, we will add more functions such as subtraction, protected devision and conditional operators to the function set. We will also investigate different ways of applying gradient descent search on inclusion factors and compare this approach with the approach of applying gradient descent search to numeric terminals only. We will also examine how the use of gradient descent of inclusion factors may affect the types of structures evolved.

References

Andre, D. (1994). Automatically defined features: The simultaneous evolution of 2-dimensional featu re detectors and an algorithm for using them. In Kinnear, K. E., editor, *Advances in Genetic Programming*, pages 477–494. MIT Press.

Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998). *Genetic Programming: An Introduction on the Automatic Evolution of computer programs and its Applications*. San Francisco, Calif. : Morgan Kaufmann Publishers; Heidelburg : Dpunkt-verlag. Subject: Genetic programming (Computer science); ISBN: 1-55860-510-X.

Howard, D., Roberts, S. C., and Brankin, R. (1999). Target detection in SAR imagery by genetic programming. *Advances in Engineering Software*, 30:303–311.

Koza, J. R. (1992). *Genetic programming : on the programming of computers by means of natural selection*. Cambridge, Mass. : MIT Press, London, England.

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

Loveard, T. and Ciesielski, V. (2001). Representing classification problems in genetic programming. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1070–1077, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea. IEEE Press.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E., McClelland, J. L., and the PDP research group, editors, *Parallel distributed Processing, Explorations in the Microstructure of Cognit ion, Volume 1: Foundations*, chapter 8. The MIT Press, Cambridge, Massachusetts, London, England.

Ryan, C. and Keijzer, M. (2003). An analysis of diversity of constants of genetic programming. In C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, E. C., editor, *Proceedings of the Sixth European Conference on Genetic Programming (EuroGP-2003)*, volume 2610 of *LNCS*, pages 404–413, Essex, UK. Springer Verlag.

Samaria, F. and Harter, A. (1994). Parameterisation of a stochastic model for human face identification. In *2nd IEEE Workshop on Applications of Computer Vision*, Sarasota (Florida). ORL database is available at: www.cam-orl.co.uk/facedatabase.html.

Song, A., Ciesielski, V., and Williams, H. (2002). Texture classifiers generated by genetic programming. In Fogel, D. B., El-Sharkawi, M. A., Yao, X., Greenwood, G., Iba, H., Marrow, P., and Shackleton, M., editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 243–248. IEEE Press.

Tackett, W. A. (1993). Genetic programming for feature discovery and image discrimination. In Forrest, S., editor, *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, pages 303–309, University of Illinois at Urbana-Champaign. Morgan Kaufmann.

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

Zhang, M. and Ciesielski, V. (1999). Genetic programming for multiple class object detection. In Foo, N., editor, *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence (AI'99)*, pages 180–192, Sydney, Australia. Springer-Verlag Berlin Heidelberg. Lecture Notes in Artificial Intelligence (LNAI Volume 1747).

Zhang, M., Ciesielski, V., and Andreae, P. (2003). A domain independent window-approach to multiclass object detection using genetic programming. *EURASIP Journal on Signal Processing, Special Issue on Genetic and Evolutionary Computation for Signal Processing and Image Analysis*, 2003(8):841–859.

Zhang, M. and Smart, W. (2004). Genetic programming with gradient descent search for multiclass object classification. In Keijzer, M., O'Reilly, U.-M., Lucas, S. M., Costa, E., and Soule, T., editors, *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 399–408, Coimbra, Portugal. Springer-Verlag.