Using Unrestricted Loops in Genetic Programming for Image Classification

Jan Larres, Mengjie Zhang, Will N Browne

***Abstract*— Loops are an important part of classic programming techniques, but are rarely used in genetic programming. This paper presents a method of using unrestricted, i.e. nesting, loops to evolve programs for image classification tasks. Contrary to many other classification methods where pre-extracted features are typically used, we perform calculations on image regions determined by the loops. Since the loops can be nested, these regions may depend on previously computed regions, thereby allowing a simple version of conditional evaluation. The proposed GP approach with unrestricted loops is examined and compared with the canonical GP method without loops and the GP approach with restricted loops on one synthesized character recognition problem and two texture classification problems. The results suggest that unrestricted loops can have an advantage over the other two methods in certain situations for image classification.**

I. Introduction

This paper is concerned with the loop construct in genetic programming (GP) for image classification. Our world is full of recurrences and many real world problems require iteration as a part of a solution. Iteration is an important feature in computer programming and provides a mechanism to execute a sequence of instructions repeatedly. This is reflected in classic programming languages such as C and Python by supporting loop constructs that allow iteration over data structures and other recurring tasks.

In terms of whether there is a fixed number of cycles to be executed for the loop body, loops can be categorised into two different types [1]. The first is the so-called *count-controlled loops*, which execute the loop body a fixed number of times and is usually implemented by either a *for-loop* or some *while-loop*s in modern programming languages. The second type is the so-called *event-controlled loops*, which repeatedly execute the loop body until a certain condition is met. In this case, the number of times for the loop body to be executed is unknown in advance, and this loop is typically implemented by the *while-loop* only. These different loops have been discussed in [1] and [2].

Regarding whether a loop can be embedded into another loop, the loops can be categorised into two types: *restricted loops*, which do not allow any loops within any other loop, and *unrestricted loops*, which allow one or more loops to occur within another loop (*nested* loops). It is important to note that our form of nesting occurs in the conditions of the loops and not the bodies as is usually the case with nested loops as the loop bodies are just simple operations.

Authors are with the School of Engineering and Computer Science, Victoria University of Wellington; email: larresjan@ecs.vuw.ac.nz, mengjie.zhang@ecs.vuw.ac.nz, will.browne@vuw.ac.nz.

A. Image Classification using GP with Loops

Image classification tasks arise in a very wide variety of practical situations. Detecting tanks and helicopters from satellite images, identifying suspected terrorists from fingerprint images and diagnosing medical conditions from X-ray images are just three examples. In many cases, performing these tasks by hand is too expensive or too slow. Given the amount of image data that needs to be classified, computer based classification programs are of immense social and economic value.

Popularised in the 1990s, genetic programming (GP) [3] employs the Darwinian principle of survival of the fittest to automatically generate computer programs for a particular task. Since the 1990s, GP has been successfully applied to image classification tasks and achieved a certain level of success [4], [5], [6].

Unlike many other classification tasks, image classification tasks often include repeated patterns. For example, within an image, the neighbouring pixels are more likely to have similar intensities. So it was considered worthwhile applying GP with loop structures to image classification.

However, compared with the standard arithmetic operators, which are commonly used in GP, loops are more complex and usually require different types of input variables and Boolean conditions. In addition, there is a great potential for infinite loops – so keeping loops valid is still challenging in GP. So the use of loops in GP is still relatively uncommon. So far, a number of papers have already applied loops to GP [1], [2], [7], but most of these works used the simple, restricted loops only. GP with unrestricted, nested loops for image classification is still rarely investigated.

When using GP approaches to image classification, a typical step is to determine a set of features to extract from the images that will then be used as terminals by the genetic programs during the evolutionary process. In this way, the features that are most likely to be useful in the classification task can be made available, thereby providing a good basis for a successful result. However, this also has some limitations: the feature set is typically problem specific and manually predefined, and it is possible that some important features are not included. It would be beneficial if the features, particularly some useful ones that may not be obvious to humans, could be automatically constructed for a particular problem by the GP evolutionary process.

B. Goals

To cope with the above issues, this paper aims to investigate a GP approach with unrestricted loops to image

978-1-4244-8126-2/10/$26.00 ©2010 IEEE

classification. Instead of using a small set of predefined image features, this approach seeks to enable good features to be automatically constructed during evolution based on image pixel values. This approach will be examined and compared with GP with restricted loops and canonical GP without loops on a number of image classification problems of varying difficulty. Specifically, we will investigate whether the new approach can do perform adequately on these image classification tasks and whether the new GP approach with unrestricted loops can outperform the other two methods.

C. Organisation

The rest of the paper is organised as follows. Section 2 briefly describes some related work to GP with loops and GP for image classification. Section 3 describes the GP system with restricted and unrestricted loops, and the related parameter settings. Section 4 describes the first test of the experiments and results by testing the proposed approach to character recognition. Section 5 presents the results of the second test of the proposed approach to two texture classification tasks. Finally, section 6 gives the main conclusions with future research directions.

II. Related Work

This section describes very briefly related work to GP with loops, GP for image classification, and GP with loops for image classification.

A. Loops in GP

The evolution of loops using GP was first explored by Koza [3], where the `Do Until (DU)` operator is introduced to allow the evolution of loops for the block stacking problem. The `DU` operator takes two arguments and appears in the form `(DU WORK PREDICATE)`. The first argument specifies the loop body, and the second specifies a predicate as a condition. To avoid infinite loops, Koza applies both a limit to the total number of loops (100) in a program and a limit to the number of cycles in each loop (25). Since the data types of the variables are not constrained effectively, the system produces many ill-formed programs causing an infinite loop.

To evolve a sorting algorithm, Kinnear used an iterative function `(dobl start end work)` [8], [9]. The system uses an `index` variable for the loop to control execution of the loop body `work`, ranging from value `start`, to the value `end`. In an attempt to avoid infinite loops, Kinnear used the same approach as Koza, but with larger limits (2000 and 200).

Langdon [10] utilised an iteration operator `forwhile` to evolve a list data structure. In his experiments, the form of the iteration is: `forwhile(s,e,l)`. This is also a counter-controlled approach, where the values of `s` and `e` specify the number of cycles in the loop. In this structure, nested loops were not allowed and the number of iterations was restricted to 32. The experiments shows that the `forwhile` provided the capability to process multiple list elements.

Recently, Ciesielski and Li used both restricted and unrestricted loops to evolve programs for controlling an agent in the modified Santa Fe Ant problem and sorting a 7-element array [11]. This work was extended later to use a variant of the for-loop in a binary classification problem [12]. In their implementation, Strongly-Typed Genetic Programming [13] is used to ensure that the generated programs are syntactically correct, and semantics is restricted to avoid infinite loops. Another experiment using only unrestricted loops on a factorial and a modified Santa Fe Ant problem was done by Chen and Zhang [1].

B. GP for Image Classification

Since the early 1990s, there has been a reasonable amount of work on applying GP techniques to image classification and object recognition. Some examples are Tackett's object classification [14], [15], Song's work on texture classification [16], [17], Ciesielski's landmark detection [18], Zhang's object classification and detection [19], [20], [4], Bhanu's work on object detection [21]. Typically, most of these works use a small set of predefined features as terminals in GP for evolution.

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 [22] and a recent book [23].

C. GP with Loops for Image Classification

There has not been much work in GP with loops for image classification tasks. A typical piece of such work is done by Wijesinghe and Ciesielski [6], where they examined the use of restricted loops for image classification tasks, using a range of greyscale images. Their goal was to determine if using restricted loops on image regions provided any advantage over single pixels, and if these loops would result in simpler programs or less overfitting. Their findings were very promising, with loops improving over programs without loops in every tested category. They also did a short test with unrestricted loops in one of their experiments, but found that these loops provided no benefit over restricted loops. An earlier work using loops that operated line-wise over images was done by Li and Ciesielski [24]. These loops did not allow nesting, and the images were relatively small.

III. The Approach: GP with Loops

Here we present the exact approach we used in our experiments.

A. Unrestricted vs Restricted Loops

We developed two different types of functions for a genetic program to access the image data: a function that returns the value of a single pixel, and a loop that performs an operation over a whole region.

The difference between the restricted and the unrestricted versions of the functions (both the point and loop types) is as follows. The restricted functions take only terminals as arguments: generated numbers for the coordinates, which

TABLE I
GP FUNCTION AND TERMINAL SETS

Method	Node Type	Name	Description
No loops	Functions	+	+ Operation
		-	- Operation
		PointR x y	Grey value at coordinate (x,y)
	Terminals	rX	A random integer X
Restricted loops	Functions	Same as for no loops, and	
		ForLoopR x_1 y_1 x_2 y_2 *Op*	Execute *Op* from (x_1, y_1) to (x_2, y_2)
	Terminals	Same as for no loops, and	
		f+	Positive sum of pixels
		f-	Negative sum of pixels
Unrestricted loops	Functions	Same as for restricted loops, except	
		PointU instead of PointR	
		ForLoopU instead of ForLoopR	
	Terminals	Same as for restricted loops	

are guaranteed to be inside the image boundaries, and an operation for the loops. This means that they can not be nested, the regions they are operating over only depend on the terminals that are created together with the program and are therefore the same for each evaluation of this program. The unrestricted versions, on the other hand, allow for the coordinates to come from function outputs as well as terminals, thereby allowing for them to nest. This poses a minor "problem": the outputs of functions are not necessarily inside the image boundaries. Thus, the unrestricted single pixel and loop functions transform those values into legal coordinates by taking them modulo the size of the image. This makes it clear that the term *unrestricted* refers to the structure of the loops and not the results since the results obviously have to be restricted to the image boundaries.

The single "pixel" functions have the following syntax:

(PointR x y) and *(PointU x y)*

where *PointR* refers to the restricted version and *PointU* to the unrestricted version, and x and y are the coordinates of a pixel to use. The function `(PointR x y)` simply returns the brightness of the pixel `(x, y)`, where x and y can only take a random number within the range of the image size (width and height). In the function `(PointU x y)`, however, x and y can come from either a particular pixel of an image, or from any other function return values including simple `(PointU x y)` and loop functions described below.

Similarly, the loop functions also have two versions:

(ForLoopR Begin End Operation)

(ForLoopU Begin End Operation)

where *Begin* and *End* are pairs of (x, y) coordinates that specify the rectangular region the loop operates over, and *Operation* is either a sum or a subtraction operation that is performed over all the pixels in the region.

For example, the loop function `(ForLoopR 1 1 2 2 f+)` will return the value of $[0+I(1,1)+I(1,2)+I(2,1)+I(2,2)]$, where `I(i,j)` is the brightness of pixel `(i,j)`, while the loop function `(ForLoopR 1 1 2 2 f-)` will return the value of $[0-I(1,1)-I(1,2)-I(2,1)-I(2,2)]$. In other words, in both the loop functions, the return values will be initialised to zero.

The image size information is used to guarantee that the evolved coordinates of the unrestricted versions lie inside their boundaries.

A possible use for the unrestricted features is tested in our experiments, where the important characteristic of the images is not always in the same position.

B. Overall Experiment Setup

In this approach, we used the tree-structure to represent genetic programs [3]. The ramped half-and-half method was used for generating programs in the initial population and for the mutation operator [25]. The proportional selection mechanism and the reproduction, crossover and mutation operators [26] were used in the learning and evolutionary process.

In particular, we used strongly typed genetic programming (STGP) [13] in experiments. STGP allows multiple data types used in a system, and enforces closure by generating parse trees that satisfy the type constraints. During the process of crossover and mutation, only sub-programs (subtrees) of the same type can be swapped or mutated. This ensures that the generated programs are syntactically correct.

This system was implemented using the RMITGP package due to its support for strongly typed GP, to ensure that only valid programs are evolved.

As mentioned earlier, this paper compares the GP with unrestricted loops method to the GP with restricted loops method and the canonical GP without using loops. The functions and terminals used in the three methods are summarised in Table I. For comparison purposes, the three methods used the same settings of the parameter values, which are summarised in Table II.

TABLE II
GP SYSTEM SETUP

GP Setting	Value
Runs	20
Population size	100
Max. generations	1000
Termination	0% error or >max gens.
System	RMITGP v1.5
Min. tree depth	1
Max. tree depth	6
Crossover rate	70%
Mutation rate	28%
Elitism rate	2%

Notice that the number of generations was intentionally set to 1000, which is larger than usual. This is because the goal here is to investigate the evolvability of the three methods rather than training time. Other parameter values are set based on the common settings and some initial trials via experiments.

For all of the three methods, the evolved genetic programs are all in a tree form and the outputs of the programs are a single floating point number. For binary classification of images, a negative program output corresponds to one class and a non-negative number to the other class. The classification error rate is used as the fitness function.

The two image classification problems used are character recognition and texture classification. The details of the two problems with a number of specific tasks and the corresponding results will be described in the next two sections.

IV. TEST 1: CHARACTER RECOGNITION PROBLEM

In the first group of test experiments, the test bed is a binary classification problem where the classifier must determine whether a given image contains either the character A or B. Figure 1 shows examples of these two classes.

This problem was derived from the task of creating CAPTCHA (Completely Automated Public Turing Test to tell Computers and Humans Apart) images [27]. CAPTCHA images are specifically designed to be hard for computers to classify, and are used to prevent malicious software from automatically filling in forms or completing other tasks.

In order to increase the difficulty of the classification, noise was added to the initial images. This prevents classifiers from using just a single pixel to distinguish between the two classes, and forces it to learn more general characteristics.

Our problem set consists of 500 100×100 pixel greyscale images, with 250 depicting the character A and 250 the character B. The pixels in each image were generated by selecting a random number from a normal distribution with a mean of 170 for the characters and 130 for the background. Both have a variance of 30 to introduce the noise.

Unrestricted loops are expected to process different regions of an image depending on previously processed ones. In order to test whether this has any measurable impact on their performance, in addition to the normal character images, several other sets were created where the characters in the images were shifted by increasing distances away from their centre position. We expect the unrestricted loops to be able to adjust their regions depending on the position of the characters and to potentially lead to better classification performance than purely random restricted loops.

Based on this idea, we made six sets *chars00, chars10, chars15, chars20, chars25, chars30*, each with 500 images, where the characters are shifted randomly by up to 0, 10, 15, 20, 25, or 30 pixels, respectively, in different directions. This allows us to determine whether the unrestricted loops perform better for certain shifting distances than for others.

These sets will be evaluated using all three methods, that is, GP without any loops, GP with restricted loops, and GP with unrestricted loops. Each experiment is repeated for 20 independent runs, and the average results are reported in the rest of this section.

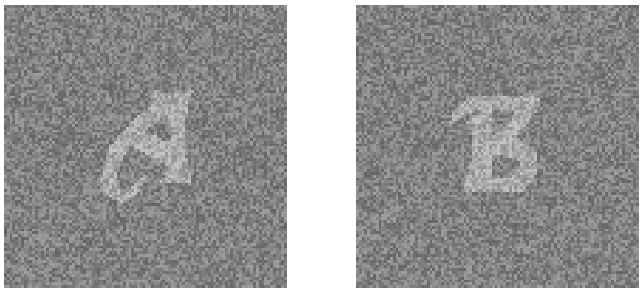


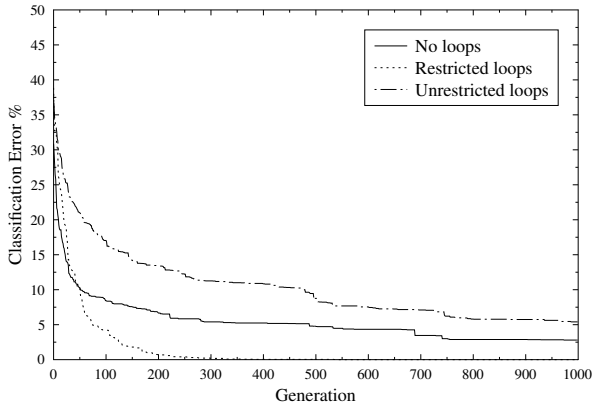
Fig. 1. Character recognition problem: example images for "A" and "B".

A. Experimental Results

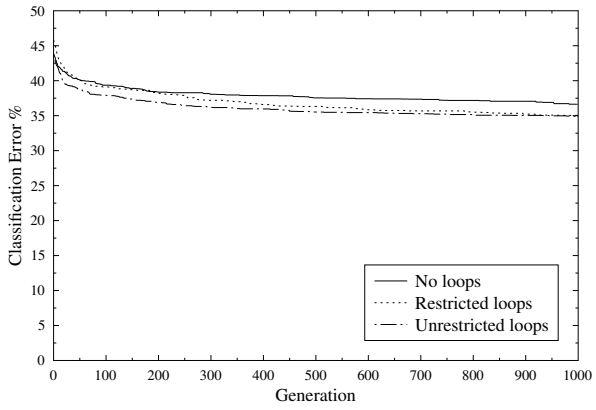
TABLE III
CHARACTER RECOGNITION: AVERAGE CLASSIFICATION ERROR

Experiment	Strategy	Classification Error rate	
		mean	std. Dev.
chars00	No loops	2.80%	2.67%
	Res. loops	0.00%	0.00%
	Unres. loops	5.40%	5.34%
chars10	No loops	32.78%	2.36%
	Res. loops	21.92%	4.14%
	Unres. loops	32.26%	2.27%
chars15	No loops	36.94%	1.72%
	Res. loops	32.04%	4.03%
	Unres. loops	35.18%	1.21%
chars20	No loops	36.62%	1.78%
	Res. loops	35.04%	2.55%
	Unres. loops	34.92%	1.75%
chars25	No loops	37.98%	1.68%
	Res. loops	36.66%	1.42%
	Unres. loops	33.84%	2.42%
chars30	No loops	37.74%	1.64%
	Res. loops	36.96%	1.72%
	Unres. loops	34.02%	1.63%

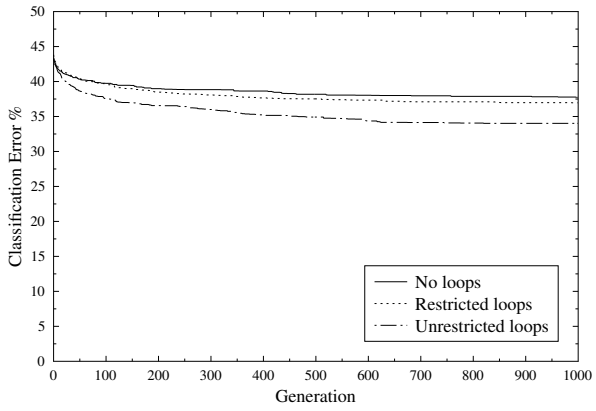
The results from our experiments are shown in Table III. Figure 2 shows the evolutionary process for the three typical tasks *chars00, chars20* and *chars30*. As can be seen from Table III and Figure 2, for the non-shifted images (*chars00*), the unrestricted loops did not perform as well as the restricted loops and were even worse than GP without any loops. This



(a) chars00: Not shifted



(b) chars20: Shifted by up to 20 pixels



(c) chars30: Shifted by up to 30 pixels

Fig. 2. Character recognition problem: Average classification error

is very likely due to the arrangements needed to keep the output of the unrestricted loops within the image boundaries so that they can be used as input in other functions, thereby inevitably reducing variety. Another possible reason is that the task without any shifting is relatively easy, but the high level features evolved by complex unrestricted loop functions cannot easily be decomposed into simple features that this simple task needs. For the task with a little shifting of *chars10*, again GP with the simple restricted loops achieved the best performance and the GP method with unrestricted loops was not as good as the restricted loops, but it performed slightly better than, or at least as well as, the GP approach without any loops.

A further check of the results for the shifted tasks reveals that with greater distances of shifting, the unrestricted loops slowly overtake the restricted loops and certainly also the GP approach without any loops. This first happens for the *chars20* task, and is apparent for *char25* and *chars30*. For these "relatively difficult" tasks, the GP method with restricted loops still achieved slight improvement over the GP method without loops, but the improvement is not statistically significant (after a T-test with the two-side 95% confidence). The new GP method with unrestricted loops, on the other hand, always performed better than the other two approaches, and the improvement was statistically significant although the amount of improvement was intuitively not large.

It is also interesting to note that the unrestricted loops almost always resulted in a lower standard deviation than the two GP methods with the restricted loops and without loops, except for the *chars00* and *chars25* tasks. In these cases the deviation is closer to the one for the no loops strategy.

These results suggest that the unrestricted loops have an advantage over restricted loops and the absence of loops if the tasks are more difficult, in this case, when the characters are shifted further from their centred position. In this case the unrestricted loops can employ their conditional nature to improve on the purely static property of the other two methods.

B. Example Evolved Programs

To check what the evolved programs look like, we give two example evolved programs here and their corresponding operated regions in the images.

The following evolved classifier program (figure 3) with unrestricted loops achieved a classification error of 14.4% on the *chars00* task. As can be seen from the Table III, this result is not good. The regions it used are shown in Figure 5a. According to this figure, although this program was reasonably successful in locating the region of interest, it has a number of functions including some loops that are relatively far from the region of interest (close to the character "A").

```
(- (PointU (+ r6 r4) (- r71 r41)) (PointU
(- r97 r79) (ForLoopU r52 (ForLoopU
(ForLoopU r88 r78 r68 r86 f-) r6 (PointU
r3 r72) r81 f+) (PointU (+ r76 r6)
(PointU r57 r15)) (PointU (ForLoopU r79
r65 r75 r29 f+) (ForLoopU r36 r71 r49 r23
f-)) f-)))
```

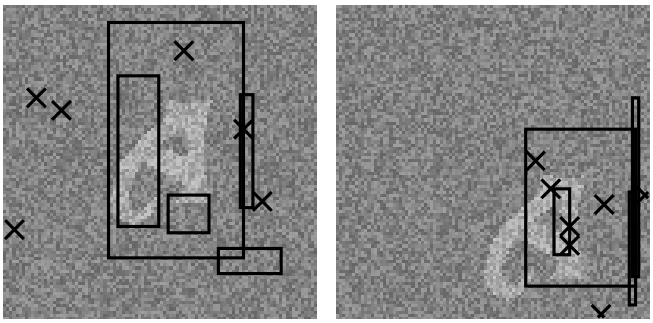
Fig. 3. Example program 1.

The following evolved unrestricted-loops classifier program (figure 4) achieved a classification error of 36.4% for

the *chars30* task. According to Table III, this result is quite close to the mean value. The regions it used in the image are shown in Figure 5b. This example shows that the loops adapt to the image content: all the regions are in the lower right area of the image where the character is found. This allows the loop to process the relevant regions of the image to make a more accurate classification.

```
(+ (+ (- (+ r5 (PointU r37 r19)) (PointU
r49 r42)) (ForLoopU (PointU r96 r22)
(PointU r49 r55) (ForLoopU r86 r92 r90
r20 f+) (ForLoopU r48 r18 (+ r0 r38) r61
f-) f+)) (ForLoopU (- (PointU r69 r99) (-
r30 r72)) r74 (+ (PointU r26 r0) (PointU
r71 r29)) (- (- r33 r64) (- r22 r16))
f-))
```

Fig. 4. Example program 2.



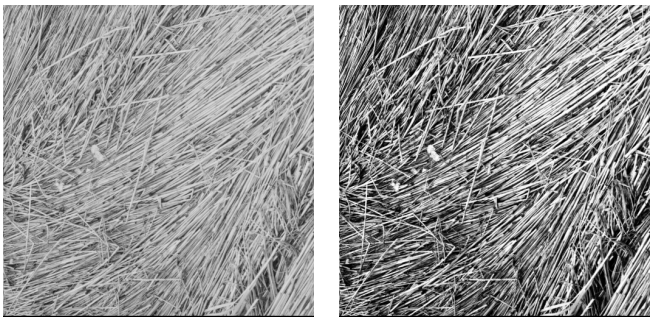
(a) Regions for chars00 experiment (b) Regions for chars30 experiment

Fig. 5. Example evolved programs and corresponding regions used.

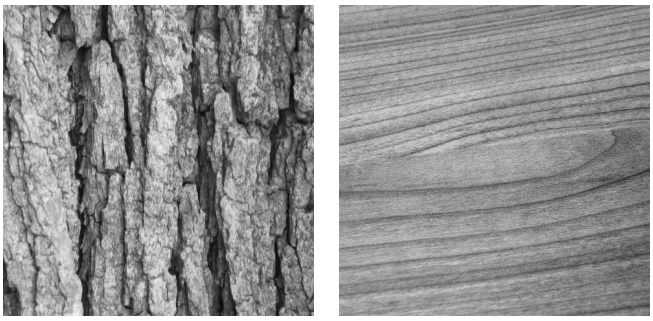
C. Further Discussion

The disadvantage of both restricted and unrestricted loops is that evaluating programs with loops takes far longer than programs without them. One run of our character recognition problem over 1000 generations took 16 minutes on average for the restricted loops and 20 minutes for the unrestricted loops, compared with only 30 seconds for the no loops strategy, but this is not unexpected. If the unrestricted loops can lead to better performance for some problems, particularly the relatively difficult problems, a longer evolutionary training time is an acceptable compromise. This can be very much offset by the advanced computer hardware such as grid computing facilities commonly used these days.

Another point is also noticed. In the previous loop approaches for non-image classification problems, such as the factorial program and the Santa Fe Ant problem [1], [2], it has been shown that the good programs evolved with loops are typically shorter than those without using loops. However, this pattern does not hold for this character classification problem here. In the next section, we will investigate two texture classification problems and examine whether this pattern can hold or not.



(a) Examples of the two classes of the Brodatz texture set



(b) Examples of the two classes of the Ponce texture set

Fig. 6. The texture problem.

V. Test 2: Texture problems

The texture problems used here comprise two different data sets of images depicting varying textures. The first one ("Brodatz Textures") comes from the USC-SIPI Image Database [28] and consists of two classes of textures, where the second class is a histogram equalized version of the images from the first class. 13 images per class are provided by this data set (512×512 pixels). Figure 6a shows an example of both classes. The second set ("Ponce Textures" [29]) comes from the Ponce research group at UIUC and consists of several different textures. Two classes are chosen from the types available, each providing 40 images (480×480 pixels). Figure 6b shows an example of both classes.

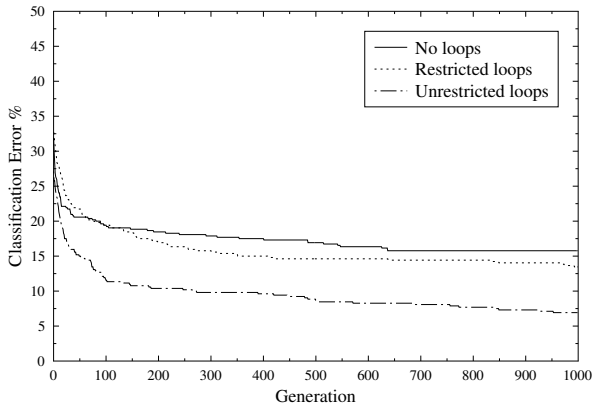
These two tasks pose two potentially more difficult machine vision problems than the character recognition problem in the previous section since it does not rely on artificial constructs, but instead on naturally occurring recurrences. The Brodatz set will also test whether histogram related differences allow the unrestricted loops to utilize their conditional nature to some advantage.

A. Experimental Results

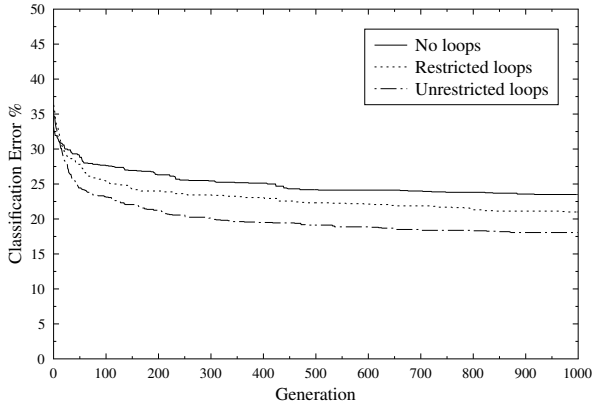
The results on the two texture problems are shown in Table IV. On the Brodatz problem, the GP method with unrestricted loops achieved the best performance among the three methods: a classification error of 6.62% for the unrestricted loops, 14.31% for the restricted loops and 15.31% for no loops. The advantage for the Ponce data set is not as large as for the Brodatz problem intuitively, but the improvements over the other two approaches are still statistically significant (by a T-test with 95% confidence). The results on both

TABLE IV
TEXTURE PROBLEM: AVERAGE CLASSIFICATION ERROR

Experiment	Strategy	Classifition Error Rate	
		mean	Std. Deviation
Brodatz	No loops	15.31%	3.12%
	Res. loops	14.31%	3.87%
	Unres. loops	6.62%	3.61%
Ponce	No loops	23.50%	3.82%
	Res. loops	21.00%	2.92%
	Unres. loops	18.07%	3.73%



(a) The Brodatz texture set



(b) The Ponce texture set

Fig. 7. Texture problem: Average classification error.

problems show a clear improvement of the unrestricted loops over the other two methods.

Figure 7 shows the progression of the average classification error over 1000 generations during evolution. These curves clearly show that among all three methods, GP with the unrestricted loops has a performance improvement over the other two methods except for the first few generations in the Ponce data set.

B. Further Discussion

Another observation is that the GP approach with unrestricted loops produced an ideal solution (all the images have been correctly classified without any false positive or false negatives) for the Brodatz set in two out of the 20 runs, while the other two methods did not produce any ideal solutions. Figure 8 shows an evolved program classifier with 0% error.

```
(- (ForLoopU (+ (ForLoopU r349 r419
r89 r255 f+) (PointU r396 r111)) r247
(ForLoopU (+ (- r277 r27) r457) (-
r333 r474) r460 (ForLoopU (PointU
r373 r379) r80 r387 r313 f+) f-) (+
r394 (+ (PointU r198 r111) r215))
f+) (ForLoopU (+ (+ r169 r463) (+
r314 r328)) (ForLoopU (+ r500 r402)
(PointU r25 r191) (ForLoopU r234
r247 r438 r36 f-) (PointU r144 r398)
f-) (- (PointU r459 r193) (ForLoopU
r84 r242 r448 r62 f-)) (ForLoopU
(ForLoopU r443 r49 r151 r109 f+)
(PointU r105 r399) (- r506 r8) r454
f-) f+))
```

Fig. 8. Example evolved program for the Brodatz problem.

Again, the ideal solution programs with unrestricted loops are quite long. Inspection of the evolved programs without any loops and those with simple restricted loops shows that they are very similar in length (all very long). This observation is quite different from the previous work on non image classification tasks. There are several possible reasons for this. One is that for the previous problems, such as the factorial and the Santa Fe Ant problems, there exists one or more perfect solutions that could be known in advance, and when designing the experiments, one can consider that *a priori* information. Another one could be that this paper used very low level features (pixels) as terminals, and the number of such terminal instances is very large. For example, for the Brodatz texture classification problems, the image size is 512×512, so the number of possible simple point terminals will be over 2.6×10^5. If we consider the fact that the unrestricted loops and points can be nested, the number will be even larger. So the potential search space is much larger, which can substantially increase the program size.

VI. CONCLUSIONS

The goal of this paper was to investigate a GP approach using unrestricted loops that can evolve better classifiers for image classification than using restricted or no loops. The goal was successfully achieved by developing an unrestricted point function and an unrestricted loop function. This approach was examined and compared with the GP approach with simple restricted loops and GP with no loops on a set of synthesized character classification tasks and two sets of non-synthesized greyscale images for texture classification.

The results show that unrestricted loops can indeed allow for an improvement in classification under certain circumstances. In our first group of experiments, the unrestricted

loops provided an advantage over the other two methods for the tasks when the characters in the images were shifted randomly from their normal positions in each image. In these situations, the two GP methods with restricted or no loops adapted less to the new position. The unrestricted loops that allowed nesting enabled the programs to perform conditional execution, providing an improvement in classification.

The non-synthesized images in our second experiment further showed a lower classification error for the unrestricted loops. The error for the first image set was only half that of the other two methods, and that for the second set also showed a clear improvement.

These results suggest that GP with unrestricted loops can outperform the GP methods with simple restricted loops and with no loops on relatively difficult problems. For relatively simple problems, GP with simple restricted loops or even with no loops can achieve very reasonable performance.

The disadvantage of loops is the increased processing time required, especially for unrestricted loops due to the increased search space. Evaluations took 10 times as long for the loop methods as the no loop method. However, as the GP with unrestricted loops can achieve significantly improvement over the canonical GP with no loops on difficult image classification tasks, using a longer duration is an acceptable compromise. This is particularly true as the computer hardware is continuing to speed up and grid computing facilities become common nowadays.

For future work, we will apply this approach to more image classification problems. We will also investigate new ways of automatically constructing high level image features using loop functions.

ACKNOWLEDGEMENT

This work is supported in part by the University Research grant at Victoria University of Wellington and the Summer Research Scholarship scheme of the New Zealand Government. Thanks also go to the GP program package *rmitgp*, which was used in the work. This package was developed by the Evolutionary Computation and Machine Learning group RMIT University, Australia.

REFERENCES

[1] G. Chen and M. Zhang, "Evolving While-Loop structures in genetic programming for factorial and ant problems," in *AI 2005: Advances in Artificial Intelligence*, 2005, pp. 1079–1085.

[2] X. Li, "Utilising restricted for-loops in genetic programming," Ph.D. dissertation, Department of Computer Science, RMIT University, 2007.

[3] J. R. Koza, *Genetic Programming: On the Programming of Computers by Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[4] M. Zhang and V. Ciesielski, "Neural networks and genetic algorithms for domain independent multiclass object detection," *International Journal of Computational Intelligence and Applications*, vol. 4, no. 1, pp. 77–108, 2004.

[5] Y. Zhang and P. I. Rockett, "Evolving optimal feature extraction using multi-objective genetic programming: a methodology and preliminary study on edge detection," in *Proceedings of the 2005 conference on Genetic and evolutionary computation*. 2005, pp. 795–802.

[6] G. Wijesinghe and V. Ciesielski, "Using restricted loops in genetic programming for image classification," in *Proceedings of the 2007 Congress on Evolutionary Computation*, Singapore, 2007, pp. 4569–4576.

[7] V. Ciesielski and X. Li, "Experiments with explicit for-loops in genetic programming," in *Proceedings of the 2004 Congress on Evolutionary Computation*, vol. 1. IEEE Press, Jun. 2004, pp. 494–501.

[8] J. Kenneth E. Kinnear, "Evolving a sort: Lessons in genetic programming," *In Proceedings of the 1993 International Conference on Neural Networks*, vol. 2, pp. 881–888, 1993.

[9] ——, "Generality and difficulty in genetic programming: Evolving a sort," *In Stephanie Forrest, editor, Proceedings of the 5 International Conference on Genetic Algorithms, ICGA-93*, pp. 287–294, 1993.

[10] W. B. Langdon, "Data structures and genetic programming," *In Peter J. Angeline and K. E. Kinnear, Jr., editors, Advances in Genetic Programming 2*, pp. 395–414, 1996.

[11] V. Ciesielski and X. Li, "Experiments with explicit for-loops in genetic programming," *In Congress on Evolutionary Computation*, pp. 494–501, 2004.

[12] X. Li and V. Ciesielski, "Using loops in genetic programming for a two class binary image classification problem," *G.I. Webb and Xinghuo Yu (Eds.): AI 2004, LNAI 3339*, pp. 898–909, 2004.

[13] D. J. Montana, "Strongly typed genetic programming. technical report bbn 7866," Bolt Beranek and Newman, Inc., Tech. Rep., March 1994.

[14] 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, 17-21 Jul. 1993, pp. 303–309.

[15] ——, "Recombination, selection, and the genetic construction of computer programs," Ph.D. dissertation, Faculty of the Graduate School, University of Southern California, Canoga Park, California, USA, April 1994.

[16] A. Song, "Texture classification: A genetic programming approach," Ph.D. dissertation, Department of Computer Science, RMIT University, Melbourne, Australia, 2003.

[17] A. Song and V. Ciesielski, "Texture analysis by genetic programming," in *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*. Portland, Oregon: IEEE Press, 20-23 Jun. 2004, pp. 2092–2099.

[18] 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*, ser. Lecture Notes in Computer Science, M. Keijzer, A. Tettamanzi, P. Collet, J. I. van Hemert, and M. Tomassini, Eds., vol. 3447. Lausanne, Switzerland: Springer, 30 Mar. - 1 Apr. 2005, pp. 351–360.

[19] 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, November 2003, pp. 402–407.

[20] M. Zhang and W. Smart, "Using gaussian distribution to construct fitness functions in genetic programming for multiclass object classification," *Pattern Recognition Letters*, vol. 27, no. 11, pp. 1266–1274, 2006.

[21] B. Bhanu, Y. Lin, and K. Krawiec, *Evolutionary Synthesis of Pattern Recognition Systems*. Springer Verlag, New York, 2005.

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

[23] S. Cagnoni, E. Lutton, and G. Olague, *Genetic and Evolutionary Computation for Image Processing and Analysis, EURASIP Book Series on Signal Processing and Communications, Vol. 8*. Hindawi Publishing Corporation, 2007.

[24] X. Li and V. Ciesielski, "Using loops in genetic programming for a two class binary image classification problem," in *Proceedings of the 2004 Australian Joint Conference on Artificial Intelligence*. Springer, Dec. 2004, pp. 898–909.

[25] W. Banzhaf, P. Nordin, R. E. Keller, and Frank D. Francone, *Genetic Programming: An Introduction on the Automatic Evolution of computer programs and its Applications*. Morgan Kaufmann Publishers, 1998.

[26] J. R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.

[27] "http://www.captcha.net/," accessed December 2009.

[28] "http://sipi.usc.edu/database/database.cgi?volume=textures," accessed in December 2009.

[29] S. Lazebnik, C. Schmid, and J. Ponce, "A sparse texture representation using local affine regions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, pp. 1265–1278, 2005.