Genetic Programming for Solving Common and Domain-independent Generic Recursive Problems

Tessa Phillips
School of Engineering
and Computer Science
Victoria University of Wellington
Wellington, New Zealand

Mengjie Zhang
School of Engineering
and Computer Science
Victoria University of Wellington
Wellington, New Zealand

Bing Xue
School of Engineering
and Computer Science
Victoria University of Wellington
Wellington, New Zealand

***Abstract*—In human written computer programs, loops and recursion are very important structures. Many real-world applications require recursion and loops. Loops and recursion can also be achieved by using genetic programming (GP). There has been a lot of work on GP for loops but not much on recursion. Our recent initial work on GP for recursion has shown that GP can be used to solve recursion problems, based on which this work develops two new GP methods that can solve a wider range of problems without decreasing the performance. The two new methods are tested on symbolic regression problems, binary classification problems, and Artificial Ant problems. They are compared to methods using loops, traditional GP, and the methods developed in our previous work. The results show that the new methods have improved the accuracy and increased the range of symbolic regression problems that the methods can perfectly solve, and improved the performance on two of the Artificial Ant problems. The new methods can also solve classification problems, and have better performance than loops on many of these tasks. This is the first work using recursion for classification problems, and is the first design of a generic recursive method for GP.**

I. Introduction

Loops and recursion are very important in human written computer programs. Iterative structures allow human written computer programs to solve repetitive tasks. Without iterative structures, human written programs would need to repeat the same code many times resulting in a program that is unnecessarily long.

Genetic Programming (GP) [1] has been used to solve a wide range of problems. A limitation with traditional GP however is that, solving large repetitive tasks is often difficult. Many real-world applications such as image classification or navigation tasks require iterations [2], [3]. Including iterative structures such as loops and recursion in GP could have the potential to work well for some of these tasks and solve this issue [4].

There has been a lot of work in using loops in GP to solve a wide range of tasks including classification tasks and navigation type problems which are real-world tasks. Loops have had very good performance on these difficult tasks [5][6][7][8].

There has not been much work done on recursion in GP, and it certainly has not been applied to any real-world tasks. The work that has been done has used a simple problem like a symbolic regression task to illustrate the method performance. Recursion has the potential to perform better than loops for many problems, or provide a more understandable solution. It is important to include this structure in GP to allow GP to better perform on a wide range of difficult problems.

Early work on recursion was done by Koza [9] developing Automatically Defined Recursion (ADR), which defined a recursive template function that could be used to create recursive functions. This is a very useful idea, however the approach in [9] was very complicated and the function itself was not easy to understand. The approach was only applied to very simple problems such as a parity problem which is a type of symbolic regression task.

Yu and Clark [10] created recursion in GP with lambda abstractions. Alexander and Zacher [11] created recursion by evolving recursive call trees. Moraglio [12] used non recursive fitness functions to evolve recursive functions.

We have done initial work on GP with recursion [13], developing three new recursive methods for GP (R1, R2, and R3), which had promising results on simple problems. The results of these methods showed that in all of the simple symbolic regression tasks and artificial ant tasks that were tested, two of our methods performed better than loops. These recursive methods provided a recursive solution to solving simple well known problems. The main limitation however is that these methods were problem specific. In addition, the previous work does not provide a solution to solving difficult or unknown tasks such as classification or image recognition with GP using recursion.

A. Aim and Objectives

The goal of this work is to develop a new general GP approach to solving problems with recursive nature. We expect the new approach to achieve better performance and obtain understandable solution programs for the problems to be solved. This works towards increasing the performance and understandability of GP on very difficult problems. The objectives to achieve this goal are two new recursive GP methods:

1) To develop a new GP method for solving common recursive problems (GPCR): The purpose of this method is to be an approach to common problems that might be similar

978-1-5090-4601-0/17/$31.00 ©2017 IEEE

to known functions. This method aims to automatically select and combine known recursive functions to solve an unknown recursive function.

The new method aims to extend the R1 method [13]. The R1 method was able to solve some simple symbolic regression problems as well as artificial ant problems, using different structures. There were not many functions given in the function set, and many common recursive functions could not be solved by this method.

2) To develop a new domain independent GP method for solving generic problems with recursive structures (GPGR): The purpose of this method is to have a general template recursive function which provides the structure for recursion but is not specific to any particular recursive problem. This will allow the method to apply across a wide range of unknown problems and allow it to solve some difficult problems such as classification and image recognition using a generic structure.

The design for this method is an extension to the R2 method [13]. The R2 method was a template recursive function and could solve simple symbolic regression and Artificial Ant problems significantly better than loops in most cases. The limitation with this method was that it could not solve general recursive problems due to restrictions in recursive structure.

II. Previous Work

We developed three recursive methods R1, R2 and R3 [13] which had varying generality.

1) R1: The first method R1 was very specific to the problems to be solved. This method had a function set that contained the model, or part of the model that was being evaluated. This method used a lot of domain knowledge as it requires at least part of the function be known and included in the function set. The way this method worked was that the GP tree would evolve to use or combine different recursive functions for each problem. This was particularly useful for the Binomial problem as this problem is a combination of factorials. The method could simply use the function that was already included in the function set to solve this more complex problem.

The problems which this method can reliably solve are problems that are simple combinations of the recursive functions from the function set. This greatly limits the range of problems that this method can solve as there are only six functions included in the function set.

2) R2: The second method, R2 defined a template function that was recursive, but the specific operators and terminals needed to be defined. This gives the recursive structure, but GP must evolve the details of the function itself.

The recursive template function is given by figure 1, which shows the structure of this function. This shows that the structure is limited to functions of the form $f(i) = i * f(i-1)$ where $*$ is a mathematical operator $(+, -, \times, /)$. This easily solves problems like Factorial, Sum, and combinations of these problems.

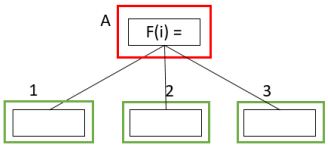


Fig. 1: General recursive structure for the R2 method. A indicates the recursive function. Input 1 is the operator for recursion $(+, -, \times, /)$. Input 2 is the base case, and input 3 is the number to evaluate the function at.

An example of this function being used is given in figure 2, which shows how for this problem, it is very simple to find a solution with the R2 function.

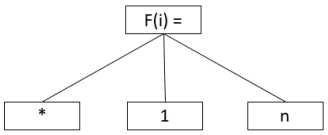


Fig. 2: Example of the R2 method for the Factorial problem. The recursive operator is $\times$, the base case is 1 and the number evaluating at is n.

This method performed well for the problems tested in [13], however there are some limitations. One limitation is that it is only for symbolic regression and artificial ant problems. The method is not general to other types of problems such as classification problems. R2 is also limited in the types of symbolic regression problems that it can solve, due to restrictions in structure. R2 cannot solve the Fibonacci problem [14], defined by $f(i) = f(i-1)f(i-2)$.

3) R3: The last method, R3 was very general, and recursion was implemented by the use of an Automatically Defined Function (ADF). This method was designed to be general, so it could solve a very wide range of recursive problems. This method worked by having the ADF include itself in its function set, which allowed the ADF to be called recursively. This meant that the recursive function did not have any restrictions in its structure.

This method did not perform as well as the other methods, and not as well as loops. The method showed some success as it was able to evolve perfect solutions on the Factorial and Sum problems, however it was difficult to evolve good solutions. The reason for this is that the R3 method is too general which makes it hard to evolve good solutions. R3 does not provide the structure for the recursive program, so it must evolve this alongside the rest of the program.

III. A New GP Method for Evolving Common Recursive Programs (GPCR)

R1 [13] aimed to solve a range of recursive problems by combining pre-existing recursive functions. The limitation of R1 was that it only solved very simple symbolic regression problems and artificial ant problems. GPCR aims to solve common recursive problems, and is designed to solve a wider range of symbolic regression problems than R1.

A. Design

GPCR aims to solve recursive problems that are either well known problems or similar to well known recursive problems. It also aims to produce very understandable solutions as the recursion is given by the existing library of recursive-functions.

The method works by using the evolutionary process to automatically search for the correct recursive functions and operators which make up the recursive program. The recursive functions are given in a pre-existing library which is included in the function set.

This new method uses a pre-existing library of 20 recursive functions to include in the function set. R1 used only 6 recursive functions. The types of function included are also different.

The new GPCR approach is designed to be limited to just symbolic regression problems. Symbolic Regression problems have well defined perfect solutions, so it is easy to find common recursive functions that are solutions for this kind of task. Other types of problems including ant problems do not have well defined perfect solutions, so this method is intentionally excluding those types of problems.

B. Recursive functions

Table I shows the recursive functions included in the pre-existing library for GPCR. This includes a range of recursive functions that depend only on $f(n-1)$, and functions that depend on both $f(n-1)$ and $f(n-2)$. This means that this method should be able to solve more complex problems, such as the Fibonacci problem as this function and some similar functions are included in the library.

The function set for all problems with the GPCR method contains the 20 recursive functions given in table I and common mathematical operators (+, - , $\times$, $\%$), where $\%$ is protected division. The terminal set contains (n, integers$\{$0-15$\}$).

Name	f(0)	f(1)	f(n)
F1	1	1	$f(n-1) \times n$
F2	0	1	$f(n-1) + n$
F3	0	1	$f(n-1) + f(n-2)$
F4	3	6	$f(n-1) \times 2$
F5	0	1	$f(n-1) + 2n - 1$
F6	2	4	$f(n-1) \times 2$
F7	2	2	$f(n-1)^n$
F8	1	2	$f(n-1) \times f(n-2)$
F9	5	7	$f(n-1) + 2$
F10	2	4	$f(n-1)^2$
F11	n	n^2	$f(n-1)^2$
F12	3	9	$f(n-1)^2$
F13	2	8	$f(n-1)^3$
F14	n	n^3	$f(n-1)^3$
F15	3	27	$f(n-1)^3$
F16	1	2	$f(n-2) \times n$
F17	1	2	$f(n-1) + f(n-2) + n$
F18	1	2	$f(n-1)^2 + f(n-2)$
F19	1	2	$f(n-1)^2 f(n-2)$
F20	1	1	$f(n-1) f(n-2) \times n$

TABLE I: Recursive functions to use for GPCR on the symbolic regression problems.

IV. A New GP Method for Solving Generic Problems with Recursive Structures (GPGR)

This method is to design a generic template recursive function which with the correct sub-trees and parameters can solve a wide range of recursive problems.

The main motivation behind this method is that, the R2 method was too restricted in the problems it could solve. The R2 method was too specific and would only be able to solve a small set of recursive tasks.

The aim of this method is to develop a general template function allows for any specific recursive function to be developed depending on the problem specified. The function should have no restrictions other than a maximum number of iterations which is used to stop the evolutionary process from being too slow.

This new design works towards creating a general recursive template function, which aims to solve many difficult symbolic regression problems and classification problems. By having a function that can be applied to all of these tasks, GP can perform well on a wide range of problems using very little domain knowledge.

A. Design of Recursive Template Function

The recursive template function aims to be a generic function that can solve any recursive based problem. This is designed such that it is not restricted in structure. The template function can evolve a recursive program of any depth and use any number of recursions within the program.

Figure 4 shows how the GPGR method works to find a solution to the factorial problem, and a more general example of how the recursive function is used is shown in Figure 3. The recursive function **A** has three inputs shown in green boxes in figure 3. The first input is the body of recursion. The second input is the base case, which defines the output at i = 0. The third input is the number that the function is being evaluated to. If input 3 had value 4 for the factorial problem shown in Figure 4, then the function would be $Factorial(4) = 4 \times 3 \times 2 \times 1$.

Inside the body of recursion, there is always a recursive terminal **B**. This terminal returns the value of a previous recursion. In the function set, two of these special terminals are included, one as in the figure that returns $F(i-1)$, and one that returns $F(i-2)$.

Another special terminal can occur within the body of recursion, which is the terminal "i". This terminal returns the current count of the recursive function. This is used in Figure 4 as the factorial function requires $Factorial(i) = Factorial(i-1) \times i$.

Strongly typed GP is used to ensure that the programs evolved are syntactically correct. This is particularly important for the recursive function and recursive terminal.

By including the recursive terminal $f(i-2)$, this method is able to solve problems like the Fibonacci problem, which is a symbolic regression problem that the previous R2 method

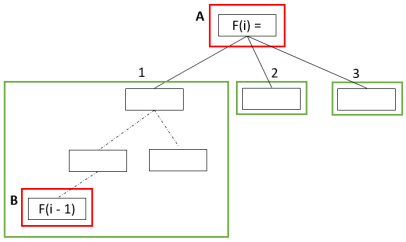


Fig. 3: General recursive structure for the GPGR method. A indicates the recursive function from the R2 method, B indicates the recursive terminal. The inputs of the recursive function are shown in green boxes and numbered from one to three.

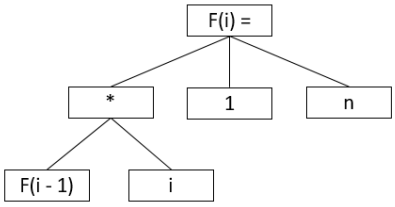


Fig. 4: Factorial(n) function as an example of the R2 structure.

could not solve. This method is also more flexible than the previous R2 method because the body of recursion is not restricted to any specific depth.

V. Experiment Design

A. Benchmark Problems

The performance of the new methods will be tested across a wide range of benchmark problems across Symbolic Regression, Binary Classification and Artificial Ant. Classification tasks use **500** training and testing instances.

1) Factorial Problem: This was used as a benchmark in [4] [13]. It is a relatively easy problem using loops and recursion, and is defined $Factorial(x) = x \times Factorial(x-1)$, $Factorial(0) = 1$.

2) Sum Problem: This function was used in [15][13] as a benchmark. It is another relatively easy problem, defined $Sum(x) = x + Sum(x-1)$, $Sum(0) = 0$.

3) Binomial Problem: The Binomial problem was used as a benchmark for recursive methods in [13]. This problem is very difficult, and is defined $Binomial(x) = \frac{Factorial(x)}{Factorial(x-k) \times Factorial(k)}$.

4) Factorial + Sum Problem: This is a problem of moderate difficulty, defined as $Factorial + Sum(x) = Factorial(x) + Sum(x)$.

5) Fibonacci Problem: This problem is a different type of recursive function from the others. The Fibonacci problem is defined as $Fibonacci(x) = Fibonacci(x-1) + Fibonacci(x-2)$ [1]. This function uses two recursions, one by x-1 and one by x-2. The base cases for this function are $Fibonacci(0) = 1$, $Fibonacci(-1) = 0$.

6) Cell: The Microscopical Cell Image Database (Serous cytology) [16][17] contains 18 different classes of cells. This task will focus on classifying between Lymphocytes non actives and Mesotheliales. The images are grey scale and have been resized to 25 $\times$ 25 pixels. To perform accurate classification, the shape of the cells is used, Loops and recursion can find the shape by using a large block of pixels.

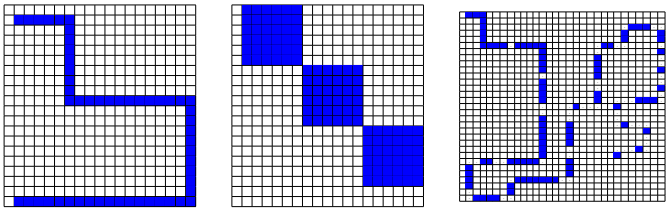


Fig. 5: Artificial Ant trails, from left: modified1, modified2, SantaFe

7) MNIST: The MNIST dataset of handwritten digits contains 28$\times$28 pixels [18]. The digits 6 and 9 were chosen to be used for a binary classification task.

8) Face: The face dataset is a collection of 19$\times$19 pixels, grey scale images of faces and non-faces [19]. The GP system must be able to recognize the key features of the face.

9) Pedestrian: The pedestrian dataset is a more challenging dataset, it contains images of pedestrians crossing roads as well as images without pedestrians [20]. The task is difficult because the images can contain background objects.

10) Poker Hand: This is a classification task to classify between single pair or an empty hand in poker [21]. The dataset has 10 attributes, which give the value and suit of each card. This is a challenging task because the relevant attributes are only every second attribute.

11) Forest Cover: The forest cover dataset is used to predict the forest cover type from 53 cartographic attributes [22]. This work uses two classes, Cottonwood/Willow and Douglas-fir.

12) MiniBooNE: The MiniBooNE dataset is used to classify between particles being electron neutrinos and muon neutrinos [21]. This is a way of classifying a signal from background. Each entry in this dataset has 50 numerical attributes.

13) Spam: The spambase dataset is used to classify emails as spam or not-spam [21]. The dataset has 57 floating point attributes that characterise the different emails.

14) Artificial Ant: The goal of the Artificial Ant Problems is too eat all the food indicated by a blue cell. The ant eats food when it moves onto a square that has food on it. Three Artificial Ant tasks were used to test the initial R1, R2 and R3 methods. The modified 1 trail [4], modified 2 trail [23], and the Santa Fe trail [23] shown in figure 5 are the three different ant trails used, and are of increasing difficulty.

B. Benchmark Methods

The following methods will be used as benchmarks for the two new methods on the benchmark problems.

1) Standard GP - No Loops: This is the standard GP method, this is not expected to do as well as loops or recursion, because the problems are very complex. The functions and terminals used for this method are {+, -, *, %, ERC(0-15)} for symbolic regression problems, and {+, -, Pixel() (or input() for non-image tasks), ERC(1-100)} for classification problems.

Parameter	Value	Parameter	value
No. Generation	50	Crossover rate	0.7
Population size	1024	Mutation rate	0.28
Initial depth	2	Reproduction rate	0.02
Maximum depth	6		

TABLE II: Showing the parameters used in the experiments.

ERC(1 - N) is a random value between 1 and N, Pixel() returns the pixel or value corresponding to the arguments.

2) Count controlled Loop - WhileLoop1: As described in [4] while loop 1 is a count controlled loop. This loop has three arguments (start, end, body). Start is the value that the loop counter starts at, end is the value it ends at, and body is the program to be executed within the loop.

Some other special terminals and functions were included. The first is a terminal i, which returns the current count value of the loop, i starts at the value of start and increases as the loop is executed until the counter is equal to the value of end.

Another special function and terminal used is SetY and Y. This is useful inside the loop as a way to store the previous value of the loop body. SetY takes in one argument and sets the value of Y to this argument.

3) Event controlled Loop - WhileLoop2: WhileLoop2 is an event controlled loop described in [4]. The loop takes two arguments the first is a condition which is of Boolean type (true or false), and the second is the loop body. The loop works by first checking if the condition is true and then executing the loop body. This loop again uses special functions and terminals i, SetY and Y.

4) R2: This is the R2 method developed in our previous work [13]. The R2 function R2() has three arguments, the first, y, is the function operator (one of $\{+, -, *, \%\}$) to use, the second is the base case, z, which is what the function returns when $x < 1$, and the third, x, is the value to use in the function. The function takes the form $f(x) = if(x < 1, xyf(x-1), z)$ where x, y, z are described above.

The function set used is the same as [13] and includes $\{+, -, *, \%, 1, 0, n, R2\}$. For the Sum problem, $\%$ is omitted as in [13], as this would allow GP to solve the problem without loops.

5) Classification Loops: For classification tasks, many different and often specific loops have been used. The classification tasks are the same as used in [5],[6], so will be compared to the results of loops in these papers as well as GP with no loops. The loops used from two papers are Li's loop, EB-Loop, Step-Loop, and Sort-Loop.

Special functions are used inside these loops, in order to aggregate the image and attribute data. These functions are sum-mode, sum and standard deviation

C. Parameter settings

The parameters used for this work were kept similar to what was done in the previous work for R1,R2,R3 in [13][5][6][23]. The typical parameters used in our work are given in table II, although there are some exceptions to these which are mentioned below.

An exception to these basic parameters was the Binomial problem, which used a population size of 2000 and a mutation rate of 0.5, cross over rate of 0.48. The change in parameters for the Binomial problem was because this problem is much harder to evolve solutions for than the others. The Binomial problem also used 150 generations. The Factorial Plus Sum problem used 150 generations, as it was a difficult problem to solve. The classification problems had a maximum depth of 8, an initial depth between 2 and 3 was used as the tasks often required a more complicated solution.

VI. Results

Table III, table IV, and table V show the results on the symbolic regression, classification, and Artificial Ant problems respectively. The results of the symbolic regression problems are compared to NoLoops, WhileLoop1, WhileLoop2 and R2. The new methods tested on these problems are both GPCR and GPGR. The classification problems are tested against NoLoops, and a range of loops for classification problems, and the only new method used for classification is GPGR. For Artificial Ant problems, GPGR is compared to whileLoop1, whileLoop2, NoLoops, R1, R2, and R3. Fitness is given by the average number of hits that the best program of each run achieves. Statistical significance was tested using a Wilcoxon rank-sum test for a significance level of 95%.

A. GPCR

The results for the GPCR method in Table III show that GPCR was able to find very good solutions for the regression tasks that were directly included in the function set. The results were not as good as for R1, due to the size of the function set increasing, making the search space larger. The GPCR method can also now solve problems like the Fibonacci problem, which is a common recursive sequence that appears in many real-world problems.

The GPCR method when compared to the benchmark loops and no loops was significantly better than them on all problems. The only exception to this was that for the Factorial + Sum problem, GPCR was similar to loop1. However GPCR had 14 perfect solutions on this method compared with none for either of the loops or no loops.

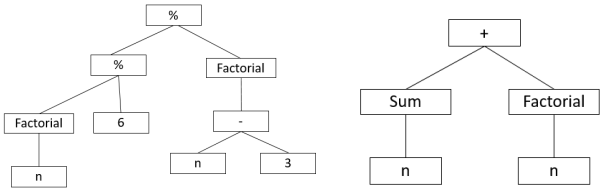
The evolved programs for the tasks were also very simple and easy to understand using knowledge about the recursive functions defined in the function set. Figure 6 shows the Binomial problem evolved program structure which is one of the more complex tasks. Figure 6 also shows the evolved program for the Factorial Plus Sum problem which is another combination of functions included in the function set. In both problems, the evolved programs are very simple, which indicates why this method performs better than benchmarks for these difficult methods, as the evolved programs are less complicated.

B. GPGR

1) Symbolic Regression Problems: Table III shows that for the factorial and sum problems. The results are similar across

Problem	Measures	No-Loop	Loop1	Loop2	GPCR	R2	GPGR
Factorial	Hits (15)	2.02±0.14	8.94±6.32[1]	7.48±6.4[1]	15.0±0.0[123]	15.0±0.0[123]	14.44±2.4[123]
	Time (ms)	184.18±21.19	331.22±1329.4	256.66±84.8	35.88±7.84	7.88±1.24	68.96±130.79
	No. Perfect (50)	0	26	21	50	50	47
Sum	Hits (15)	1.74±0.44	10.86±6.05[1]	8.56±6.21[1]	15.0±0.0[123]	15.0±0.0[123]	14.76±1.68[123]
	Time (ms)	120.82±18.68	403.04±940.71	443.92±453.69	34.68±5.9	7.62±0.77	158.08±461.09
	No. Perfect (50)	0	34	24	50	50	49
Binomial	Hits (15)	1.58±0.6	3.08±0.59[1]	2.66±0.65[1]	5.82±2.9[123]	4.54±2.77[123]	5.44±3.3[123*]
	Time (ms)	316.86±62.47	256141.16±1145089.66	78094.54±149008.64	1596.78±571.63	630.66±169.43	3468.86±2199.75
	No. Perfect (50)	0	0	0	4	3	3
Factorial + Sum	Hits (15)	2.16±0.37	2.44±0.5[1]	2.18±0.38	5.84±5.74[13]	2.26±0.44	3.64±3.38[123*]
	Time (ms)	481.46±58.22	390508.14±853500.48	789.56±230.41	1173.88±410.51	446.14±123.94	585.0±183.47
	No. Perfect (50)	0	0	0	14	0	4
Fibonacci	Hits (15)	4.52±0.54	4.74±1.56	4.04±0.2	13.24±4.03[123]	4.68±0.9[3]	8.18±4.74[123*]
	Time (ms)	495.32±53.83	218894.8±874728.91	804.2±600.02	252.94±306.81	468.04±115.23	514.32±284.79
	No. Perfect (50)	0	1	0	42	0	16

TABLE III: Table showing results on **Symbolic Regression** problems. [1] indicates significantly better than no loops, [2] indicates significantly better than WhileLoop1, [3] indicates significantly better than whileLoop2. * indicates that the GPGR method is significantly better than the R2 method.



(a) Binomial (b) Factorial Plus Sum

Fig. 6: Best evolved programs for the Binomial and Factorial Plus Sum problems using GPCR method.

all the recursive methods, and the loops also perform well. The Binomial problem proves to be difficult for all the methods, but the GPGR method is significantly better than the loops, no loops and the R2 method. The Fibonacci and Factorial + Sum problems are of medium difficulty and have good results with the GPGR method, where GPGR is significantly better than the benchmarks for the two problems. The Fibonacci problem did not get any perfect solutions with the R2 method, which is not surprising as the method cannot handle two recursive calls within the same function, which is what the solution requires.

The results show that GPGR has improved in performance over the R2 method on the more difficult symbolic regression tasks. The performance has decreased on the simple (Factorial and Sum) symbolic regression problems. This is due to the new method being more general than the previous. The R2 method was very well suited to the Factorial and Sum problems so it had nearly perfect performance on them. The new method requires a larger program tree to perfectly solve these tasks which makes it more difficult. The GPGR method still performs well on these simple problems, as it is still able to find perfect solutions in most cases. The new method however can now solve tasks like the Fibonacci problem and other problems of that form. The results for the Binomial problem show that the new method is an improvement over the previous, however it is still only able to get a few perfect solutions on this problem. A possible reason for this is that the search space is not very smooth. The way that the fitness function works for these tasks is that it counts the number of hits the problem is getting with a maximum of 15. There are about four fitness cases that usually hit, but the other 11 are more difficult. This causes GP to evolve many programs that are far from the perfect solution but that hit 4 or 5 of the fitness cases.

2) Classification Problems: GPCR was also used for classification tasks. The results were compared against that of GP with loops that were developed specifically for classification tasks, these results are shown in table IV. The results show that the R2 recursive method was significantly better than many of the loops on some of the tasks, and at least better than one of the loops on all tasks. A problem that was well suited to this recursive structure was the Poker Hand problem. The method however did no perform as well as the step loop for this problem. The reason this was suited to the GPGR method was the $f(i-2)$ recursive terminal. The solution was able to skip over every other attribute, as the poker hand task has attributes representing the value and then the suit of each card in a poker hand. The task was to classify between a single pair and an empty hand, therefore the suit attributes were irrelevant, so the GPGR method skipped over these. The step loop also has this property of being able to skip over every other attribute, but requires a slightly shallower GP tree which is probably why the performance was slightly better for this loop than our recursive GPGR method. The best program for the poker hand dataset had 99.8% training accuracy, and 99.6% testing accuracy. This is a very good performance, however the step loop was able to find many perfect solutions with 100% testing and training accuracy. This indicates that there is likely some added complexity in the GPGR method that makes it harder to find a good solution quickly.

GPGR had very good performance on two of the image problems when compared to the other methods, the GPGR method was significantly better than all of the benchmarks on the Face dataset, and GPGR was significantly better than all but one benchmark for the Cell dataset. For accurate classification on the cell dataset, the classifier needs to evaluate the shape of the cells. It was expected to have high performance with recursion on the cell problem, as it has had good performance with loops in prior work. Loops and recursion allow a classifier to use large blocks of pixels as well as single pixels.

Dataset	Measures	No-Loop	EB-Loop	Li's Loop	Step - Loops	Sort-Loop	GPGR
Forest	Training accuracy %	78.35±2.4	77.05±2.44	80.15±2.02	84.08±2.54	66.85±1.56	82.69±2.6^{1235}
	Testing accuracy %	67.93±4.64	69.69±5.66	74.96±6.23	79.19±6.95	62.62±2.81	75.22±7.17^{125}
	Time (s)	4.49	47.58	50.59	144.53	65.03	164.48
Poker	Training accuracy %	57.55±0.82	56.69±0.83	71.11±2.14	100.0±0.0	84.95±0.76	94.42±3.84^{1235}
	Testing accuracy %	52.64±2.6	52.65±1.87	66.14±2.01	99.93±0.18	87.56±0.96	92.22±4.91^{1235}
	Time (s)	4.32	24.14	59.69	12.77	46.26	188.78
MiniBooNE	Training accuracy %	82.96±0.22	82.7±0.53	84.71±1.43	88.72±1.95	87.85±1.54	88.32±1.21^{123}
	Testing accuracy %	75.86±0.8	75.89±1.13	78.71±1.34	81.8±2.24	83.68±1.93	81.54±1.76^{123}
	Time (s)	5.08	37.73	62.92	149.83	76.8	261.91
Spam	Training accuracy %	87.65±4.92	88.31±2.13	91.41±1.47	91.2±1.63	72.78±1.56	85.25±4.21^{5}
	Testing accuracy %	86.58±4.09	86.88±2.49	89.43±1.35	89.19±1.9	74.31±1.58	85.04±3.65^{5}
	Time (s)	4.72	42.04	46.46	67.66	57.14	92.42
Mnist	Training accuracy %	99.26±0.47	98.83±0.85	99.52±0.14	99.46±0.21	66.37±0.64	98.97±0.37^{5}
	Testing accuracy %	96.66±0.73	97.95±1.3	98.87±0.42	98.46±0.63	61.68±1.03	96.19±0.9^{5}
	Time (s)	11.11	155.46	73.32	434.82	376.3	82.1
Pedestrian	Training accuracy %	83.08±1.04	80.6±1.6	84.58±2.29	82.05±1.5	82.35±2.58	82.8±0.87^{24}
	Testing accuracy %	77.3±1.85	76.75±1.95	78.12±3.02	76.56±2.26	78.38±3.28	77.06±2.02
	Time (s)	8.55	169.63	169.63	363.34	268.28	91.55
Face	Training accuracy %	96.36±0.91	93.5±2.23	96.11±1.65	95.77±2.06	84.99±2.82	97.89±1.06^{12345}
	Testing accuracy %	91.1±1.69	87.86±2.74	90.69±2.7	91.56±3.07	86.71±2.43	94.4±1.53^{12345}
	Time (s)	7.8	95.08	31.38	300.28	285.27	451.06
Cell	Training accuracy %	95.85±0.43	95.17±2.06	96.14±0.59	95.92±1.26	89.95±0.86	96.74±0.55^{12345}
	Testing accuracy %	93.84±1.25	92.68±2.41	94.02±0.91	93.42±1.96	86.58±1.39	94.3±0.49^{2345}
	Time (s)	9.48	106.59	50.17	437.03	243.77	202.01

TABLE IV: **Classification** results showing the GPGR method against the benchmarks. Statistical significance of the GPGR method is indicated with ^{1} for statistically better than no loops, ^{2} for statistically better than EB-Loop, ^{3} for statistically better than Li's Loop, ^{4} for statistically better than the Step-Loop and ^{5} for statistically better than the Sort-Loop

3) Artificial Ant Problems: The results for GPCR on Ant problems are given in table V. These results show that for the Artificial Ant problems. The new GPGR approach has significantly better performance on the difficult trails, modified 2 and Santa Fe. The results on the simplest trail, modified 1, show that the R2 method performed better than the new approach. The new approach however, was still able to find many perfect solutions to this simple trail.

The results for the GPGR method showed that, on the modified 1 trail, which is the easiest trail, the performance was not as good as R2, and not as good as whileloop2. The reason for this is that with the event based loop or recursion like in R2, the ant is able to follow the trail of food. It does this by moving within the loop, and then adjusting direction so that the ant is always facing in the direction of food. With a count based approach, there would need to first be a check for food ahead and then the same loop as in the event based loop. For the other two problems, the results are better than R2, however they still are not able to find perfect solutions. These problems require very complex solutions to solve them, including nested recursion for modified 2. Overall the GPGR method has improved results on the two most difficult out of three tasks.

Figure 7 shows the best evolved programs trails for the GPGR method. This shows that for the modified 1 problem, the ant follows the correct path entirely. The modified 2 problem, the best trail (with 95/108 food eaten) traverses most of the available grid and still misses some food. The Santa Fe trail was only able to get 45/89 of the food for that trail, but this result shows that it is following a large portion of the trail, however travels a lot of unnecessary paths.

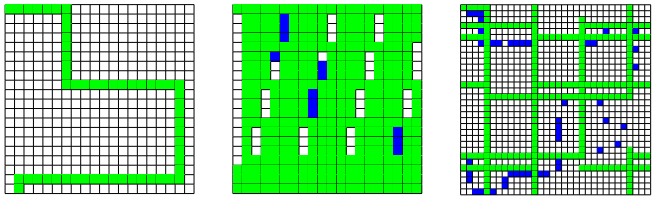


Fig. 7: Trails of best ant problems from left to right, modified 1, modified 2, Santa Fe. Blue indicates the original trail that the ant has missed, green indicates where the ant has traveled.

C. Analysis of Evolutionary Process

The average of the best-of-generation fitness from 50 runs was also examined for each of the symbolic regression and Artificial Ant problems. The classification problems were not tested for this, as the benchmark loop results were taken from previous work. For the symbolic regression problems, GPCR, and R1 would both converge to a high fitness very quickly. GPCR was similar to R2 and slightly faster than Loops and No loops. For the Artificial Ant problems, GPGR would converge at a similar rate to loops, and R2. GPGR was also still improving at the final generation, so this indicates that it could perform better if it was run for more generations. The graphs showing these trends are available at homepages.ecs.vuw.ac.nz/~xuebing/Supervison/489_2016Tessa_Phillips_FinalReport.pdf.

Problem		No-Loops	Count loop	Event Loop	R1	R2	R3	GPGR
Modified1	Food Eaten (53)	21.84±5.71	39.06±14.23[1]	45.34±10.16[1]	52.9±0.7[123]	53.0±0.0[123]	24.3±17.43	43.54 ± 12.07 [126]
	Time (ms)	330.2±52.59	223.3±112.59	270.32±171.43	96.0±158.37	26.68±22.33	370.62±109.36	1328.56 ± 1458.63
	No. Perfect (50)	0	21	27	49	50	9	28
Modified2	Food Eaten (108)	26.18±5.32	41.92±16.78[1]	36.56±9.43[1]	108.0±0.0[123]	36.84±5.87[1]	31.12±17.62	46.74 ± 16.83[1356]
	Time (ms)	360.76±70.89	358.6±81.59	2060.16±6391.75	39.2±44.39	415.42±57.71	451.68±64.41	2765.02 ± 2144.97
	No. Perfect (50)	0	0	0	50	0	0	0
SantaFe	Food Eaten (89)	13.2±4.13	16.5±7.5[1]	15.44±5.94[1]	89.0±0.0[123]	13.04±8.78	14.64±8.66	18.62 ± 7.38[1356]
	Time (ms)	362.58±68.12	306.22±39.06	343.74±51.52	147.64±120.72	440.74±100.78	393.86±58.42	11440.8 ± 45968.02
	No. Perfect (50)	0	0	0	50	0	0	0

TABLE V: **Artificial Ant** results show the GPGR method against the benchmarks. Statistical significance is indicated with [1] for significantly better than no loops, [2] significantly better than loop1, [3] significantly better than loop2, [4] significantly better than R1, [5] significantly better than R2, [6] significantly better than R3.

VII. Conclusions

This work aimed to develop a new general GP approach to solving problems with recursive nature. This goal was successfully achieved by extending R1 and R2 to develop two new methods, GPCR and GPGP. The new methods were evaluated against nine state of the art algorithms on 16 different problems of four different types.

GPCR was intended as a recursive method to solve common recursive symbolic regression problems. This method extended R1 by increasing the size of the function set to include a wider range of common recursive problems. The results of this work showed that GPCR was able to perform very well on a range of common recursive problems.

GPGR was intended as a generic method for solving recursive problems. GPGR extended the R2 method by making the structure much more general. GPGR now allows for multiple recursive calls and a recursive body of an arbitrary depth. The results show that for symbolic regression problems, the GPGR method was able to find perfect solutions to all problems tested, including problems that the R2 method could not solve. GPGR performed significantly better than loops for all the symbolic regression problems. For classification problems, GPGR was able to perform consistently well across a wide range of image and non-image binary classification problems. GPGR was better than the majority of the benchmark loops for classification. The results of GPGR on Artificial Ant problems showed that the performance was significantly better than the R2 design for the two harder trails, and still able to find many perfect solutions for the easiest trail.

References

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

[2] A. Lensen, H. Al-Sahaf, M. Zhang, and B. Xue, "Genetic programming for region detection, feature extraction, feature construction and classification in image data," in *European Conference on Genetic Programming*, vol. 9594, pp. 51–67, Springer International Publishing, 2016.

[3] M. Iqbal, B. Xue, and M. Zhang, *Reusing Extracted Knowledge in Genetic Programming to Solve Complex Texture Image Classification Problems*, pp. 117–129. Springer International Publishing, 2016.

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

[5] F. Abdulhamid, S. Andy, K. Neshatian, and M. Zhang, "Evolving genetic programming classifiers with loop structures," *2012 IEEE Congress on Evolutionary Computation*, 2012.

[6] F. Abdulhamid, K. Neshatian, and M. Zhang, "Genetic programming for evolving programs with loop structures for classification tasks," *The 5th International Conference on Automation, Robotics and Applications*, 2011.

[7] J. Larres, M. Zhang, and W. N. Browne, "Using unrestricted loops in genetic programming for image classification," *IEEE Congress on Evolutionary Computation*, 2010.

[8] D. MCGAUGHRAN and M. ZHANG, "Evolving more representative programs with genetic programming," *International Journal of Software Engineering and Knowledge Engineering*, vol. 19, no. 01, pp. 1–22, 2009.

[9] J. R. Koza, *Genetic programming III: Darwinian invention and problem solving*, vol. 3. Morgan Kaufmann, 1999.

[10] T. Yu and C. Clark, "Recursion, lambda-abstractions and genetic programming," *Cognitive Science Research Papers-University Of Birmingham CSRP*, pp. 26–30, 1998.

[11] B. Alexander and B. Zacher, *Parallel Problem Solving from Nature – PPSN XIII: 13th International Conference, Ljubljana, Slovenia, September 13-17, 2014. Proceedings*, ch. Boosting Search for Recursive Functions Using Partial Call-Trees, pp. 384–393. Cham: Springer International Publishing, 2014.

[12] A. Moraglio, F. E. Otero, C. G. Johnson, S. Thompson, and A. A. Freitas, "Evolving recursive programs using non-recursive scaffolding," in *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pp. 1–8, IEEE, 2012.

[13] T. Phillips, M. Zhang, and B. Xue, "Genetic programming for evolving programs with recursive structures," *To appear in: IEEE Congress on Evolutionary Computation*, 2016.

[14] G. Darvas, "Fibonacci numbers in nature," *Symmetry: Cultural-historical and ontological aspects of science-arts relations The natural and man-made world in an interdisciplinary approach*, pp. 109–130, 2007.

[15] S. Silva, J. Foster, M. Nicolau, P. Machado, and M. Giacobini, *Genetic Programming: 14th European Conference, EuroGP 2011, Torino, Italy, April 27-29, 2011, Proceedings*. LNCS sublibrary: Theoretical computer science and general issues, Springer, 2011.

[16] O. Lezoray, A. Elmoataz, and H. Cardot, "A color object recognition scheme: application to cellular sorting," *Machine Vision and Applications*, vol. 14, pp. 166–171, 2003.

[17] O. Lezoray, "Microscopical cell image database (serous cytology)." [Online]. Available: https://lezoray.users.greyc.fr/researchDatabasesSerousCells.php.

[18] Y. LeCum and C. Cortes, "The mnist database of handwritten digits." [Online]. Available: http://yann.lecun.com/exdb/mnist/.

[19] M. cbcl, "Face data," 2000. [Online]. Available: http://cbcl.mit.edu/software-datasets/FaceData2.html.

[20] S. Munder and D. M. Gavrila, "An experimental study on pedestrian classification," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 11, pp. 1863–1868, 2006.

[21] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml.

[22] S. Hettich and S. D. Bay, "The uci kdd archive," 1999. [Online]. Available: http://kdd.ics.uci.edu.

[23] X. Li, *Utilising restricted for-loops in genetic programming*. PhD thesis, School of Computer Science and Information Technology, Faculty of Applied Science, Royal Melbourne Institute of Technology, Melbourne, 2007.