Relative Referenced Genetic Programming

John M. Palmer
Istituto di Metodologie per l'Analisi Ambientale (IMAA),
Consiglio Nazionale delle Ricerche, C.da S. Loja Z.I.,
85050 Tito Scalo, Potenza, Italia

This paper presents a linear code referencing approach to the representation of individuals within a genetic programming scheme. This approach has been devised in order to confront various problems associated with genetic programming schemes. These are primarily the size of the available search space, the ability to pass through this search space, the construction of valid individuals after crossover and mutation, and the probability for the use of terminals and subpieces of an individual's solution. A comparison is made with existing methods and for the problems tested the presented method gives the best results.

1. Introduction

The automatic production of mathematical expressions or algorithms is considered desirable when the nature of a problem is uncertain but there is some idea about the variables involved. Genetic programming [1] is an evolutionary method introduced by Koza in 1990 [2] and is an extension of optimization with genetic algorithms [3]. Genetic programming evolves solutions to problems by using a population of possible solutions and applying evolutionary operations such as selection, crossover, and mutation. Genetic algorithms can operate in both discrete and continuous search spaces depending on how the solution is represented and modified. One of the main advantages of genetic programming, and often a key factor for its use in the wider sciences, is that the output solutions can usually be interpreted analytically and separated into functions. This is in contrast to other optimization methods that work with sets of coefficients such as artificial neural networks [4], where the solution is not so intuitive.

Relative referenced genetic programming (RRGP) can be considered an extension of multi-expression programming (MEP) [5], however the method has been developed independently and the use of references that are not entirely connected to the gene position make it fundamentally different. The aim of this paper is to introduce a new approach for the representation of individuals in a genetic programming scheme. The first part of this paper will describe the representation, then this will be tested with some numerical problems. These problems have

been taken from previous work comparing linear genetic programming techniques [6].

2. Representation

The representation of RRGP individuals is split into chromosomes and genes. There are three types of chromosomes and the individual is read along the gene position. The first chromosome type (reference chromosome) holds reference labels, the second (function chromosome) selects the function to be applied to inputs, and the third type (action chromosome) controls the movement of reference labels. The third chromosome type is optional and not used in the basic RRGP mode. In RRGP there is a fixed number of terminals and these take reference labels. These reference labels become reassigned to outputs during execution of the individual. An output may have multiple references. Consider the example in Table 1 for the simple expression $(a+b)/c$ in the basic RRGP mode. In the basic RRGP mode all contributing references to a function's input are reassigned to the function's output, and only two reference chromosomes are used. The individual (sometimes called the genome) is interpreted sequentially along the genes. Note that after the first calculation $(a+b)$ the output is assigned both reference labels *1* and *2*. This has the implication that if the last reference in chromosome A was *2* then the individual would produce the same output. An RRGP individual in this basic mode will construct a solution in pieces. Because references group to outputs there is an increased probability to combine the larger pieces of a solution. A population will search the space of a solution, biased toward the work that has already been done in previous generations. The search space is limited by the number of references; however, all individuals are valid. Now consider another example for the expression $\sqrt{(a^2+b^2)}$, as shown in Table 2. Note that gene four could be written in 24 possible ways and eight of these will give the desired output. Of course for some expressions a terminal needs to be

Gene Position	1	2
Reference Chromosome A	*1*	*1*
Reference Chromosome B	*2*	*3*
Function Chromosome	1	2
Output expression	$a+b$	$(a+b)/c$
Output's references	*1* and *2*	*1*, *2*, and *3*

Table 1. An example representation for the function $(a+b)/c$. Terminals: a, b, c. Reference labels assigned to terminals: $a \to 1$, $b \to 2$, $c \to 3$. Function set labels: $(inputA + inputB) \to 1$, $(inputA/inputB) \to 2$. The individual is encoded as the values shown in the chromosomes. The gene position and outputs are shown here so the evaluation of an individual can be better understood.

Gene Position	1	2	3	4
Reference Chromosome A	*1*	*2*	*1*	*1*
Reference Chromosome B	*1*	*2*	*2*	*1*
Function Chromosome	3	3	1	5

Table 2. A possible solution for the expression $\sqrt{(a^2 + b^2)}$. Terminals: a, b. Reference labels assigned to terminals: $a \to 1$, $b \to 2$. Function set: $(A + B) \to 1$, $(A - B) \to 2$, $(A * B) \to 3$, $(A/B) \to 4$, $\sqrt{A} \to 5$, $\sqrt{B} \to 6$.

Gene Position	1	2	3	4	5	6	7	8
Reference Chromosome A	*1*	*1*	*3*	*2*	*2*	*1*	*1*	*1*
Reference Chromosome B	*1*	*1*	*3*	*2*	*4*	*3*	*2*	*6*
Function Chromosome	3	3	3	3	3	1	1	1

Table 3. An example representation for $a^4 + b^3 + a^2 + b$ demonstrating the use of terminal repeats. Terminals: a, b. The number of terminal repeats is set to 3. Reference labels assigned to terminals: $a \to 1$, $b \to 2$, $a \to 3$, $b \to 4$, $a \to 5$, $b \to 6$. Function set: $(A + B) \to 1$, $(A - B) \to 2$, $(A * B) \to 3$, $(A/B) \to 4$.

used a number of times; two solutions to this are (*a*) the introduction of the action chromosome, which is discussed later, or (*b*) terminals are repeated in the initial referencing. In the basic configuration terminals are simply repeated. Since it would not always be possible to predict how much one terminal would be used compared to another, all terminals are repeated by the same amount. A search can be biased toward the construction of the smaller pieces of a solution by making many terminal repeats. Consider the expression $a^4 + b^3 + a^2 + b$; an example solution using terminal repeats is given in Table 3. Here the relative nature of the representation can be seen more clearly, not only because the references represent outputs from previous calculations, but also because the operation of a gene is not strongly dependent on the gene position. Equally the position of a gene does not strongly affect the output of the following genes. Note that the genes in Table 3 could be shuffled around in a number of ways without affecting the result. This is also a very important factor affecting the crossover and mutation operators and making a wider variety of operations possible such as a shuffle mutation, removal of genes, or inexact crossover. For clarity, note that references gradually group together so an output may have a number of references. For example, the reference in position eight of reference chromosome A could be *1*, *2*, *3*, or *4* equivalently.

Adding the action chromosome deviates from the basic RRGP model but should be considered as it allows the space of possible solutions to be searched differently. Here we consider that for each operation, the choice of action to be taken for the contributing references, should also be part of the solution, hence coded in a separate chromosome. The

Gene Position	1	2	3	4	5	6	7	8
Reference Chromosome A	*1*	*1*	*3*	*2*	*2*	*1*	*1*	*1*
Reference Chromosome B	*1*	*1*	*3*	*2*	*4*	*3*	*2*	*4*
Function Chromosome	3	3	3	3	3	1	1	1
Action Chromosome	1	1	1	1	2	1	1	1

Table 4. An example representation for $a^4 + b^3 + a^2 + b$ demonstrating use of the action chromosome. Terminals: a, b. The number of terminal repeats is set to 2. Reference labels assigned to terminals: $a \to 1$, $b \to 2$, $a \to 3$, $b \to 4$. Function set: $(A + B) \to 1$, $(A - B) \to 2$, $(A * B) \to 3$, $(A/B) \to 4$. Reference action: *All contributing references to the input now reference the new output* $\to 1$. *All references to inputA now reference the new output* $\to 2$. *All references to inputB now reference the new output* $\to 3$.

expression from the previous example $a^4 + b^3 + a^2 + b$, may be coded as shown in Table 4. This is mostly the same as the previous example, the difference is that the output from gene five does not take reference *4*, or any that would be grouped with it. Reference *4* continues to reference the terminal b, until gene eight. This way fewer terminal repeats are needed.

3. Fitness calculation

Each gene of an RRGP expression produces an output, for this reason there are multiple outputs to be tested. Simply, an individual's fitness [1, 7] is the fitness calculated from the best output. There are two ways to approach this. Typically a number of test cases are used to assess an individual, therefore an individual's fitness may be considered as the average fitness over all test cases, where the best output is used in each case. This option may be written as

$$Q_A = \frac{1}{N} \sum_{j=1}^{N} \min_{g=1\ldots C} \{q_{gj}\} \tag{1}$$

where Q_A is the fitness of the individual, there are N test cases, and an individual has a chromosome length of C genes. The fitness of the output for test case j at gene position g is q_{gj}. Alternatively an individual's fitness may be considered as the fitness of the best gene averaged over all test cases. This second fitness assessment option may be written as

$$Q_B = \min_{g=1\ldots C} \left\{ \frac{1}{N} \sum_{j=1}^{N} q_{gj} \right\}. \tag{2}$$

It is worthwhile implementing both of these options as they encourage different behavior in the evolving individuals and so the type of solution may be different. The first option Q_A, encourages more subsolutions

to be formed and allows for the final solution to be the best from a selection of outputs. This may cause a problem because no information is included as to how to select a particular output, given a particular input; a classification algorithm would be needed after the solution is found. However in most cases where a comprehensive training set is used, if there is a solution with a single output then this is likely to be in a state of lower potential. The term "potential" is used to describe the state of a solution as constrained by the representation, genetic operators, selection and fitness function, and that solutions will most likely change in a direction of decreasing potential. This may be considered analogous to thermodynamics including work and entropy. Using Q_B encourages the coalescing of solutions within individuals, this is beneficial for a general solution but may limit the early search by decreasing the space in a population to test subsolution pieces. Some problems may benefit from both these measures and so a combination may be used. One idea may be to gradually change from one measure to the other or more simply the average of the average fitness measures may be used,

$$Q_C = \frac{Q_A + Q_B}{2}. \tag{3}$$

Note that in this model a minimization of the fitness is considered so smaller fitnesses represent solutions that produce outputs closer to the objectives. If the genetic programming (GP) model were to maximize fitness then equivalently the minimum operators in equations (1) and (2) would need to be maximum operators. The fitness function for any output measured against a test case is independent of the RRGP representation and is chosen at the programmer's discretion. Here is a distance measure normalized with the magnitude of the target value:

$$q = \left\{ \frac{|o - t|}{1 + |t|} \right\}^a \tag{4}$$

where o is the output value, t is the target value, and a is a constant used for fitness scaling [8], which is normally set to 1.

4. Selection and genetic operators

RRGP is a method of representation, it does not prescribe the type of GP model to be used [9], nor what selection of genetic operators [1, 10, 11] to use. These are considerations of the programmer and depend on the problem to be solved. To test the RRGP representation with the possible methodology would take many hours of work and is beyond the scope of this paper. This is not to say that the RRGP representation does not have implications to genetic operators, and, as mentioned previously, in fact increases the possibilities. The main reason why there is more

Gene Position	1	2	3	4
Reference Chromosome A	*1*	*2*	*1*	*1*
Reference Chromosome B	*1*	*2*	*2*	*3*
Function Chromosome	3	3	1	4

Table 5. An example coding $(a^2 + b^2)/c$ that demonstrates the effects of swapping around adjacent genes. Terminals: a, b, c. Reference labels assigned to terminals: $a \to 1$, $b \to 2$, $c \to 3$. Function set: $(A + B) \to 1$, $(A - B) \to 2$, $(A * B) \to 3$, $(A/B) \to 4$.

freedom to test genetic operators is because of the way a solution is interpreted *via* the references, to the extent that an individual could be read backwards and would still produce a valid solution, although possibly not a very good one. Secondly, subparts of a solution are not directly fixed to the gene position. To demonstrate this, a simple type of shuffle mutation is described, where a gene and the following gene are swapped around. Consider the example in Table 5 using the basic RRGP model.

Table 5 gives the expression $(a^2 + b^2)/c$. If the genes in positions two and three are swapped then the individual codes $(a^2 + b)^2/c$. If instead gene positions three and four are swapped then the expression becomes $(a^2/c) + b^2$. Now note that as written earlier, the reference in position four of reference chromosome A could also be *2* and the same expression would be coded. However, this time if the last two genes are swapped the output expression would become $(b^2/c) + a^2$. Swapping the first two genes would not alter the expression, but would change the effect of further operations. Searching the space of solutions using RRGP can be more morphological than can be expected with standard GP techniques.

5. Example test problems

Three test problems are used to demonstrate this referencing technique. These are taken from a paper by Oltean and Groşan [6], where a variety of GP techniques are compared. Figure 1 shows the results (from [6]) of these test problems in terms of generation and success rate over 100 runs. The test problems are labeled T_1, T_2, and T_3:

$$T_1, \quad f(x) = x^4 + x^3 + x^2 + x$$
$$T_2, \quad f(x) = \sin(x^4 + x^2)$$
$$T_3, \quad f(x) = \sin(\exp(\sin(\exp(\sin(x))))).$$

Oltean and Groşan test different linear GP representations by fixing the GP model, and variables such as population size and mutation rate, and then changing the representation for a set of problems. However, this is not so easily done because different representations have different algorithmic complexities. To obtain the results given in Figure 1, the GP

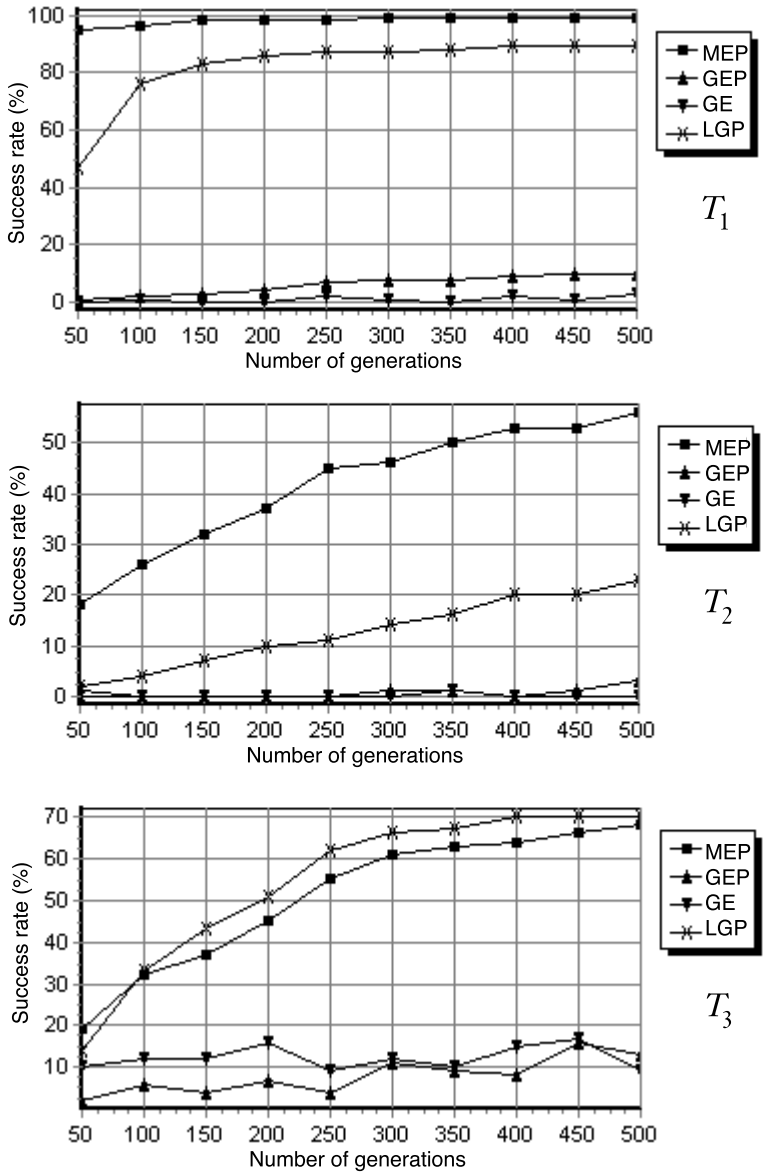


Figure 1. The relationship between the success rate and number of generations for the test problems T_1, T_2, and T_3. This figure is taken from [6] and compares the different GP techniques: multi-expression programming (MEP), gene expression programming (GEP), grammatical evolution (GE), and linear genetic programming (LGP).

models were adjusted in order to consider the algorithm complexity for the different representations. Equivalently for the RRGP configuration this would mean a population of 100 individuals and a chromosome length of 20 genes. The chromosome length is decided upon by considering the number of symbols in an individual's representation but does not consider the degrees of freedom of these symbols. Note then that comparing algorithm complexity in this manner can only be used as a guide. The functions used are +, −, ∗, /, sin, and exp.

The RRGP representation is tested against the results in Figure 1 by considering the equivalent GP model and variables for each of the test problems. Because the comparable model is not so simply defined, further testing is made around the chosen model to give some idea of how the results are affected by the algorithm's complexity. In addition to this, some of the tests are made to further demonstrate the behavior of the RRGP representation. For simplicity, it is noted for each test problem which results are considered most comparable with those in Figure 1.

5.1 Algorithm configuration

For all experiments, the algorithm's configuration is set as follows unless otherwise stated in the text or figures.

- Number of terminal repeats: 5.
- Terminals: x.
- Function set: $(A + B)$, $(A - B)$, $(A * B)$, (A/B), $\sin(A)$, $\exp(A)$.
- Length of chromosome: 20 genes.
- Population: 100.
- Reference action options:
  All references to inputA now reference the new output → 1.
  All references to inputB now reference the new output → 2.
- Fitness calculated as Q_B from equation (2).
- Probability for a symbol to be randomly mutated: 0.05.
- Crossover rate: 0.9.
- Elite count: 2.
- Selection method: binary tournament.
- A run is considered successful and stopped when the fitness of the best individual is less than 1.0×10^{-5} for three successive generations, the successful generation is taken to be the first of these three.

The results for the RRGP testing show the cumulative frequency of success over 100 runs, this is equivalent to the success rate since the range is 0 to 100. As in [6], each individual's fitness is evaluated using 20 random test cases where x is taken from the range 0 to 10. For the experiments presented in this paper a test set pool of 2000 cases is randomly generated at the start of each run, then at the start of each generation 20 of these cases are randomly selected to evaluate each individual.

Note that all the runs were made using a standard home laptop computer with an Intel Core2 CPU T5500, with 2GB RAM. However, for the sake of algorithm performance, the main consideration is the number of calculations made per run. For GP algorithms most calculations are made by the evaluation of individual solutions. Therefore, for one run, the important factors are the mean number of calculations per individual, the population size, and the number of generations. For the RRGP representation, the mean number of calculations per individual can further be split into calculations per gene.

5.2 Test problem 1

The first problem T_1, is relatively simple to solve as can be seen from the results. Figure 2 shows the results for T_1 for various population sizes and a reduced function set of $(A + B)$, $(A - B)$, $(A * B)$, and (A/B). The chromosome length is also limited to 10 genes. These first results are to demonstrate the ability of the RRGP representation and how the reduced function set affects the complexity of the problem. It should be noted that even though a population of 100 individuals reaches 100% success rate in the fewest generations, a population of 30 or 50 individuals reaches 100% success rate with fewer individual evaluations.

To make a better comparison to the results from Figure 1, the algorithm is configured as described in section 5.1, and the results are shown in Figure 3. Various changes to the configuration are also shown in the figure, and the behavior can be seen to be much the same. In one of the tests a test set pool of 20 cases is used, this way the same cases are chosen for each generation. This is done because the way a problem is presented to a learning system can considerably change the way the system learns the problem and the quality of the solution. For this problem the different presentation of the data shows no apparent change in the results. Another of the tests for problem T_1 gives only one option for the action chromosome, namely: *All references to inputA now reference the new output*. This way it is not important to encode the chromosome in the individual and so it can be removed. Additionally a test is made with a chromosome length of 10 genes which reduces the gene evaluations per individual and so processing time. Reducing the number of genes also reduced the available search space and the ability to move through that space, however no notable difference

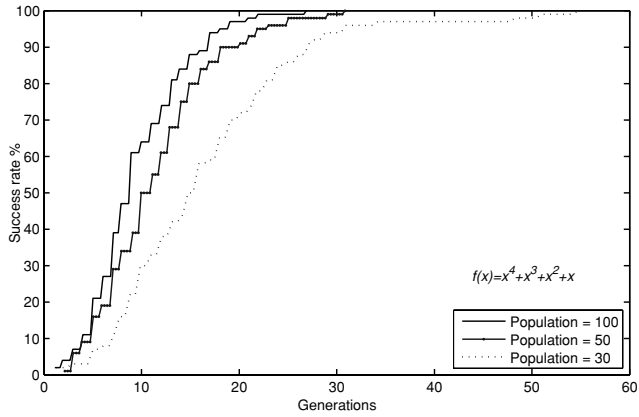


Figure 2. The relationship between success rate and generations for the test problem T_1 using the RRGP representation with a reduced function set of $(A + B)$, $(A - B)$, $(A * B)$, and (A/B). Tests for three population sizes are shown.

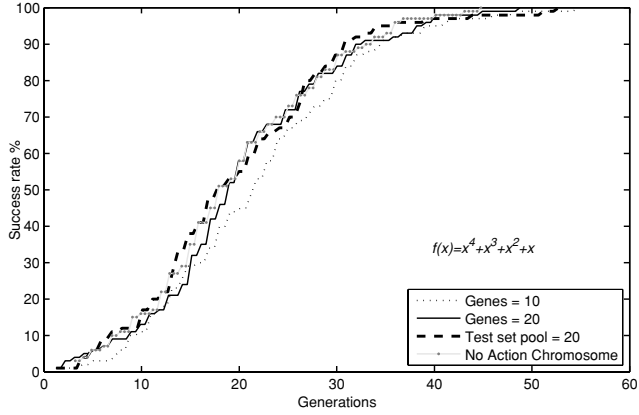
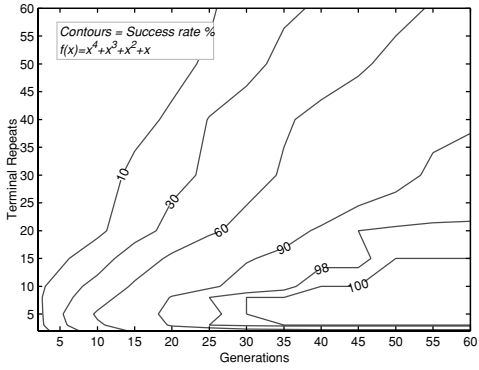


Figure 3. The relationship between success rate and generations for the test problem T_1 using the RRGP representation with the extended function set of $(A + B)$, $(A - B)$, $(A * B)$, (A/B), $\sin(A)$, and $\exp(A)$. Four variations of the configuration are shown. The plot marked with the solid black line represents the GP model most comparable with the one used to produce the data in Figure 1.

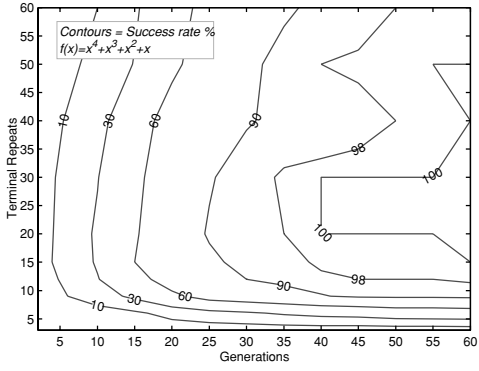
can be seen in the results. This is a very positive result given that the GP model comparable with the results in Figure 1 has 20 genes.

As described in section 2, the action chromosome and number of terminal repeats can considerably affect the way the problem space is searched. Here, the terms open and closed search spaces will be used. A

closed search space is one with definite bounds as enforced by the search representation irrespective of chromosome length. An open search space is expandable with algorithm/chromosome length. Note that in the RRGP representation in the basic mode, the search space can be easily closed by treating genes as valid only where each reference chromosome references a different terminal or output. In the basic RRGP representation however, different reference chromosomes are allowed to reference the same thing, these can be considered as opening/expansion points in the representation. The RRGP representation with the inclusion of the action chromosome can therefore be seen to increase the number of opening points. A representation with more opening points will be less constrained but more prone to bloat. Figure 4 shows how by changing



(a)



(b)

Figure 4. Contour plots showing the relationship between terminal repeats, generations, and success rate for test problem T_1 with the reduced function set of $(A + B)$, $(A - B)$, $(A * B)$, and (A/B). A comparison is shown between (a) the RRGP representation in the basic mode and (b) including the action chromosome and so changing the reference action options.

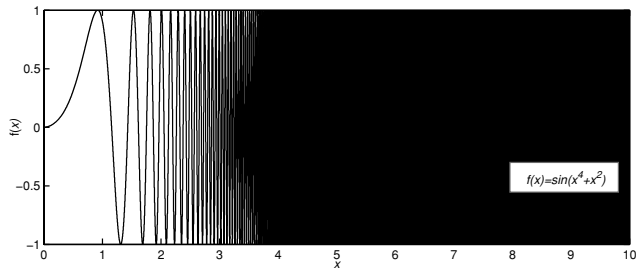


Figure 5. The function for test problem T_2 in the range $0 \leq x \leq 10$.

the search space the success rate of a problem can be altered; T_1 is again used with the smaller function set. The success rate is measured every five generations for a maximum of 60 generations over 100 runs. This is repeated for a range of terminal repeats, first in the basic mode and then by including the action chromosome and so changing the reference action options. The results are plotted as contour plots.

5.3 Test problem 2

The second problem T_2, is the hardest to solve but is interesting from the point of view of data sampling for the test cases. Genetic algorithms and GP techniques build solutions in an approximate manner starting with large features and gradually moving toward the finer detail. However the details of a solution do not always result in an equivalently smaller error. Consider 20 random points taken for the inputs 0 to 10 for the problem function of T_2, which is plotted in Figure 5 for this range. It can be seen that for this problem, changing an approximate solution only slightly can have a misleading effect on the errors. A solution to this for future work would be to consider the stability of a test case in respect to the solution being tested, and then use this to weigh the test cases.

The results after implementing the RRGP representation are shown in Figure 6. Because of the lower success rates, 500 generations are shown. As explained earlier, the major limitations to producing good success rates for this problem lie in the data sampling and the expression of the data to the algorithm. This is demonstrated by making another test of 100 runs but this time sampling 20 points from the range $0 \leq x \leq 2$, the plot can also be seen in Figure 6. As originally presented this problem is badly sampled and increasing the range in x becomes almost like adding noise to the problem.

5.4 Test problem 3

Finally, T_3 gave good results as can be seen in Figure 7. Notice that the results in the basic RRGP mode are better than with the extra action

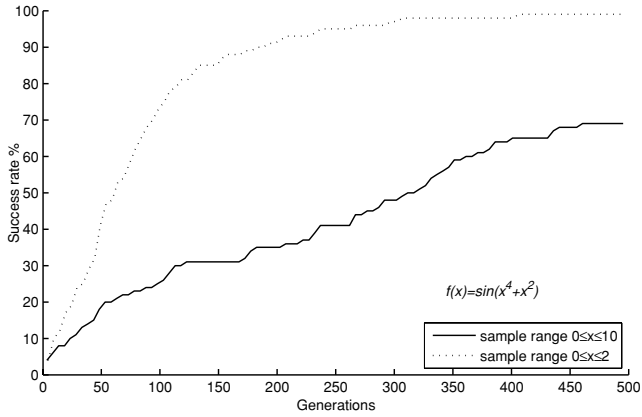


Figure 6. The relationship between success rate and generations for the test problem T_2 using the RRGP representation. Two sample ranges for the test cases are shown. Compare these ranges with Figure 5. The solid black line is most comparable with the results shown in Figure 1.

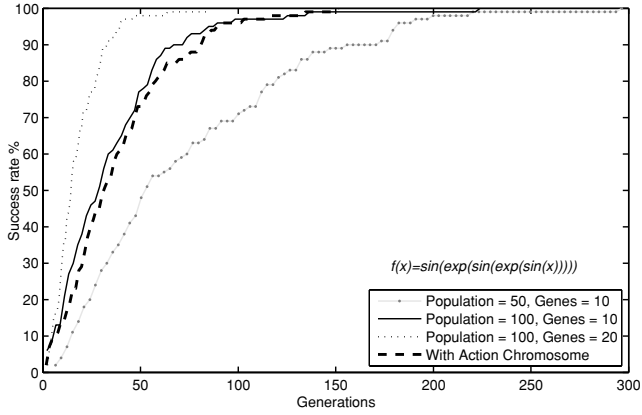


Figure 7. The relationship between success rate and generations for the test problem T_3 using the RRGP representation. Four variations of the configuration are shown. Notice that where the action chromosome is included the population is 100 and the individuals have 20 genes, this is shown with the thick dashed line. The plot shown with the dotted line, representing a population of 100 and 20 genes is most comparable to the plot in Figure 1.

chromosome options. This problem has a tree representation that is deep instead of wide—it is likely this which makes it easier in the basic RRGP mode with a few terminal repeats (in fact the problem only needs one repeat to be representable).

6. Some advantageous features of relative referenced genetic programming

6.1 Bloat control

In the basic RRGP representation the search space is constrained, references group to outputs and so the evolution of individuals is protected against bloat [12, 13], where solutions grow in size without efficiently increasing fitness. When an action chromosome is introduced the search space can be made less constrained, however with a tendency to group references bloat is dampened. The topic of bloat and introns, though much discussed, is incomplete and with what may seem quite different points of view. The definition of an intron is not always the same; [13] uses the terms "inviable code" and "unoptimized code" to be more specific. A clear definition of an intron would be any code that does not contribute to a final solution given all possible test cases. This only includes unoptimized code if the solution is exact for all possible test cases. This definition considers generalization, and so what may be considered introns (in the sense of not being useful) in the unoptimized code cannot be any more known than the optimal solution. Take the example of the Pythagoras problem to evolve an expression to calculate the hypotenuse of a triangle given the other two sides x and y. If for all the test cases x and y were similar then it is likely $\sqrt{(x \times x) + (x \times x)}$ evolves before the general solution. This could be optimized to $\sqrt{2x^2}$ but in hindsight is best left alone. Introns are formed as a result of the evolutionary process and in turn can have both positive and negative effects on further evolution. In the RRGP representation, as with linear GP, introns can also take the form of code disconnected from the final solution [4]. Genes that cause exceptions in the RRGP representation are not removed and so may also be considered introns. The different types of intron, how they should be defined, and how they contribute to bloat, remain topics of much discussion.

6.2 Exception handling

Exception handling is easy with the RRGP representation. If a gene causes an exception it can simply be omitted from the interpretation. Information is not lost and the fact that an exception occurred is represented by the fitness of the individual. Expressions automatically and morphologically restructure to omit the failing operation.

6.3 Chromosome length

The order in which references appear is more influential to a solution than simply gene position. Bloat can be dampened. Valid individuals can be easily constructed. There is more freedom in the use of genetic

operators. For these reasons RRGP could be used for a variable length chromosome GP model. This has not been tested and is left for future work.

6.4 Scope to experiment

Many variations such as the following can be tested within this referencing representation beyond that of the genetic operators.

- Include a second function chromosome which takes only values 1 or 0, and turns a gene on or off. This can be easily tested without having to implement much more code. Preliminary tests of this have shown that solutions can be reached in less time.
- The action chromosome could also direct outputs to a temporary memory location that would have its own reference which could not be grouped.
- It is possible to use more than two reference chromosomes, moving away from the binary tree structure. It should be noted that the function chromosome would select the reference chromosome's inputs to use, that is to say a function $f(A, B, C) = A + B$ is separate from the function $f(A, B, C) = A + B + C$.
- Additional types of references could be tested. For example, a reference to the most referenced output.
- Additional action chromosome options could be tested. For example, an option to reset references to their original terminals.

6.5 Repeating patterns in expression representations

Some solutions for expressions are more interesting than others, consider the expression x^{16} in the basic RRGP model shown in Table 6. This shows how the search space can be constrained, in fact, here with only one terminal, four functions, and three gene positions there are only 64 possible expressions. By increasing the terminal repeats to 2, expressions such as x^5 can be coded, and x^{16} could be coded in a number of different ways. Another interesting example is the symbolic regression $x^4 + x^3 + x^2 + x$ shown in Table 7. Only the first three genes are unique, then the genes in positions two and three repeat. If they continued to repeat the regression would continue to extend with additional powers of x.

Gene Position	1	2	3
Reference Chromosome A	*1*	*1*	*1*
Reference Chromosome B	*1*	*1*	*1*
Function Chromosome	3	3	3

Table 6. Example coding of x^{16}. Terminals: x. The number of terminal repeats is 1. Reference labels assigned to terminals: $x \to 1$. Function set: $(A + B) \to 1$, $(A - B) \to 2$, $(A * B) \to 3$, $(A/B) \to 4$.

Gene Position	1	2	3	4	5	6	7
Reference Chromosome A	*2*	*1*	*2*	*1*	*2*	*1*	*2*
Reference Chromosome B	*1*	*2*	*3*	*2*	*3*	*2*	*3*
Function Chromosome	4	1	3	1	3	1	3
Action Chromosome	3	3	3	3	3	3	3

Table 7. Example coding of $x^4 + x^3 + x^2 + x$. Terminals: x. The number of terminal repeats is 3. Reference labels assigned to terminals: $x \to 1, x \to 2, x \to 3$. Function set: $(A + B) \to 1$, $(A - B) \to 2$, $(A * B) \to 3$, $(A/B) \to 4$. Reference action: *All contributing references to the input now reference the new output* $\to 1$. *All references to inputA now reference the new output* $\to 2$. *All references to inputB now reference the new output* $\to 3$.

7. Conclusions and suggestions for future work

Relative referenced genetic programming (RRGP) provides a new and different approach by which a number of problems in the field of genetic programming can be analyzed. The importance of gene order over absolute gene position, the constraints on search spaces, a fixed genetic palette whilst at the same time reusing code, are all features of RRGP that can be contrasted with present approaches and used to aid wider investigation into the functioning of genetic programming.

In comparison to the results in [6], RRGP performs very well producing better results than four other genetic programming methods, for all the tests made in this paper. RRGP still requires much testing and comparison with other techniques and for a wider variety of problems.

Some representations for expressions show repetition of genes, an interesting project would be to test more intelligent mutation operations that use the occurrences found within a population. This could work much like texture synthesis [14, 15] and inpainting [16] in image processing, constructing either completely new individuals or mutating parts of chromosomes. Alternatively loops could be encoded in the individuals so genes can be evaluated a number of times. This should be carefully controlled so the interpretation of an individual does not become too long.

Considering the difficulties in test problem T_2, a system could be tested where the contribution to fitness test cases are weighted.

Using a hybrid technique by measuring fitness with Q_A from equation (1), then performing a classification may be an effective solution for real-world problems.

Acknowledgments

I would like to thank family, friends, and colleagues for their support. Also I would like to thank Andrew Tarpey for his helpful suggestions and the unknown referee for their constructive comments.

References

[1] J. R. Koza and R. Poli, "Introductory Tutorials in Optimization, Search and Decision Support: A Genetic Programming Tutorial," Chapter 8, edited by E. Burke (2003) (available from http://www.genetic-programming.com/jkpdf/burke2003tutorial.pdf).

[2] J. R. Koza, "Genetically Breeding Populations of Computer Programs to Solve Problems in Artificial Intelligence," Computer Science Department, Stanford University, Stanford, CA 94305 USA (1990).

[3] M. Mitchell, *An Introduction to Genetic Algorithms* (MIT Press, Cambridge, MA, 1996).

[4] M. Brameier and W. Banzhaf, "A Comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining," in *IEEE Transactions on Evolutionary Computation*, **5**(1) (2001) 17–26.

[5] M. Oltean and C. Groşan, "Evolving Evolutionary Algorithms using Multi Expression Programming," in *The Seventh European Conference on Artificial Life*, edited by W. Banzhaf *et al.*, LNAI 2801, 651–658 (Springer-Verlag, Berlin, 2003).

[6] M. Oltean and C. Groşan, "A Comparison of Several Linear Genetic Programming Techniques," *Complex Systems*, **14** (2003) 285–313.

[7] K. E. Kinnear, Jr., "Fitness Landscapes and Difficulty in Genetic Programming," in *Proceedings of the 1994 IEEE World Conference on Computational Intelligence*, volume 1 (IEEE Press, Orlando, Florida, USA, 27–29 June 1994).

[8] V. Kreinovich, C. Quintana, and O. Fuentes, "Genetic Algorithms: What Fitness Scaling is Optimal?" *Cybernetics and Systems*, **24**(1) (1993) 9–26.

[9] F. Vavak and T. C. Fogarty, "A Comparative Study of Steady State and Generational Genetic Algorithms for Use in Nonstationary Environments," in *Evolutionary Computing: AISB Workshop, Lecture Notes in Computer Science*, volume 1143 (Springer, 1996).

[10] R. Poli and W. B. Langdon, "On the Search Properties of Different Crossover Operators in Genetic Programming," in *Proceedings of the Third Annual Genetic Programming Conference*, edited by J. Koza *et al.* (Morgan Kaufmann, Madison, WI, USA, July 1998).

[11] W. B. Langdon, "Size Fair and Homologous Tree Genetic Programming Crossovers," in *Proceedings of the Genetic and Evolutionary Computation Conference*, edited by W. Banzhaf *et al.* (1999).

[12] W. B. Langdon and R. Poli, "Fitness Causes Bloat: Mutation," in *Proceedings of the First European Workshop on Genetic Programming, Paris*, edited by W. Banzhaf, R. Poli, M. Schoenauer, and T. Fogarty (Springer-Verlag, Berlin, 1998).

[13] S. Luke, "Code Growth Is Not Caused by Introns," in *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference (GECCO-2000)* (2000).

[14] A. Efros and T. Leung, "Texture Synthesis by Non-Parametric Sampling," in *Proceedings of the International Conference on Computer Vision*, volume 2 (September, 1999).

[15] H. Igehy and L. Pereira, "Image Replacement through Texture Synthesis," in *Proceedings of International Conference on Image Processing*, volume 3 (October, 1997).

[16] M. Bertalmio, G. Sapiro, V. Caselles and C. Ballester, "Image Inpainting," in *Computer Graphics (SIGGRAPH '00 Proceedings)*, edited by K. Akeley (2000).