A New Crossover Technique for Cartesian Genetic Programming

Genetic Programming Track

Janet Clegg
Intelligent Systems Group,
Department of Electronics
University of York, Heslington
York, YO10 5DD, UK
jc@ohm.york.ac.uk

James Alfred Walker
Intelligent Systems Group,
Department of Electronics
University of York, Heslington
York, YO10 5DD, UK
jaw500@ohm.york.ac.uk

Julian Francis Miller
Intelligent Systems Group,
Department of Electronics
University of York, Heslington
York, YO10 5DD, UK
jfm7@ohm.york.ac.uk

ABSTRACT

Genetic Programming was first introduced by Koza using tree representation together with a crossover technique in which random sub-branches of the parents' trees are swapped to create the offspring. Later Miller and Thomson introduced Cartesian Genetic Programming, which uses directed graphs as a representation to replace the tree structures originally introduced by Koza. Cartesian Genetic Programming has been shown to perform better than the traditional Genetic Programming; but it does not use crossover to create offspring, it is implemented using mutation only. In this paper a new crossover method in Genetic Programming is introduced. The new technique is based on an adaptation of the Cartesian Genetic Programming representation and is tested on two simple regression problems. It is shown that by implementing the new crossover technique, convergence is faster than that of using mutation only in the Cartesian Genetic Programming method.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming—*Program synthesis*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods and Search

General Terms

Algorithms, Design, Performance

Keywords

Cartesian Genetic Programming, optimization, crossover techniques

1. INTRODUCTION

Koza [6, 7] introduced Genetic Programming (GP) in 1992. He used tree structures as the representation of the members of the population and suggested a crossover technique in which random sub-branches of the parent tree structures are swapped to produce the offspring. This sub-tree crossover was, at the time, thought to be the dominant operator within the optimization process: responsible for exploiting existing genetic material in searching for better solutions. However, it has since been found [1, 8, 9] that this sub-tree crossover technique does not always perform well. Angeline [1] compared the performance of sub-tree crossover with a crossover technique which simply mutated the sub-branches of the trees. It was found that the difference between the performances of sub-tree crossover and that of simply mutating the sub-branches was statistically insignificant. This result implied that, in some cases, sub-tree crossover was no better than some simple mutation of the sub-branches. Luke and Spector [8, 9] also compared sub-tree crossover with a simple mutation of the branches of the trees over a range of problems. They also concluded that sub-tree crossover performed little better than a simple mutation of the branches. Due to findings like these, some people now implement their GP's without using crossover at all, i.e. using mutation only.

By contrast, in Genetic Algorithms (GAs) mutation is considered to be a background operator and of secondary importance to the crossover operator. GAs have been extremely successful when applied to many real life complex optimisation problems [3, 2, 4]. Although mutation is an important genetic operator in the GA, the crossover operator contributes a great deal to its performance. Much work has been done in analysing the effects of crossover and mutation on the performance of a GA [5, 15, 17]. In [5], Jong presents experimental results illustrating the power of crossover and in [14] Schaffer compares mutation and crossover in a GA and concludes that mutation alone is not always sufficient. The inspiration for the work in this paper has been to find a new crossover technique in Genetic Programming which can contribute to the performance of the GP as much as crossover operators contribute to the performance of a GA.

Recently, Miller and Thomson [11, 12] introduced a new form of GP called Cartesian Genetic Programming (CGP), which uses directed graphs to represent programs rather than the more traditional representation of programs as trees. The CGP is implemented with mutation only and has not, up to the present time, used a crossover technique. Even so, it has been shown that the CGP performs better

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

than the traditional GP. The work described in this paper is based on this CGP representation.

This paper introduces a new method for crossover in Genetic Programming which improves the performance of the GP by speeding up its convergence considerably. The new technique has been developed based on the Cartesian Genetic Programming representation described above. The CGP representation is modified in order to enable the new crossover technique to be applied. Crossover when applied to a CGP using the traditional representation hinders its performance rather than improves it, and this has been the motivation for introducing the new representation here. The new method of crossover has been tested on two simple regression problems and the results show that it successfully speeds up the convergence of the CGP for these problems. Section 2 of this paper describes the traditional CGP method and Section 3 shows how crossover techniques fail when the CGP is in its traditional integer representation. Section 4 introduces the new representation and crossover and Section 5 describes the regression problems which the new crossover is tested on. Section 6 reports the results of using the new technique on these regression problems and finally Section 7 discusses conclusions and future work.

2. CARTESIAN GENETIC PROGRAMMING (CGP)

Cartesian Genetic Programming is a form of Genetic Programming (GP) invented by Miller and Thomson [12], for the purpose of evolving digital circuits. However, unlike the conventional tree-based GP [6], CGP represents a program as a directed graph (that for feed-forward functions is acyclic). The benefit of this type of representation is that it allows the implicit re-use of nodes in the directed graph. CGP is also similar to another technique called Parallel Distributed GP, which was independently developed by Poli [13]. Originally CGP used a program topology defined by a rectangular grid of nodes with a user defined number of rows and columns. However, later work on CGP showed that it was more effective when the number of rows is chosen to be one [19]. This one-dimensional topology is used throughout the work we report in this paper.

In CGP, the genotype is a fixed length representation and consists of a list of integers which encode the function and connections of each node in the directed graph. However, the number of nodes in the program (phenotype) can vary but is bounded, as not all of the nodes encoded in the genotype have to be connected. This allows areas of the genotype to be inactive and have no influence on the phenotype, leading to a neutral effect on genotype fitness called neutrality. This unique type of neutrality has been investigated in detail and found to be extremely beneficial to the evolutionary process on the problems studied [12, 19, 16].

Each node is encoded by a number of genes. The first gene encodes the node function, whilst the remaining genes encode where the node takes its inputs from. The nodes take their inputs in a feed forward manner from either the output of a previous node or from the program inputs (terminals). Also, the number of inputs that a node has is dictated by the arity of its function. The program inputs are labelled from 0 to $n-1$, where n is the number of program inputs. The nodes encoded in the genotype are also labelled sequentially from n to $n+m-1$, where m is the user-defined bound for the number of nodes. If the problem requires k program outputs, then k integers are added to the end of the genotype, each encoding a node output in the graph where the program output is taken from. These k integers are initially set as the outputs of the last k nodes in the genotype. Figure 1 shows a CGP genotype and corresponding phenotype for the function $x^6 - 2x^4 + x^2$ and Figure 2 shows the decoding procedure.

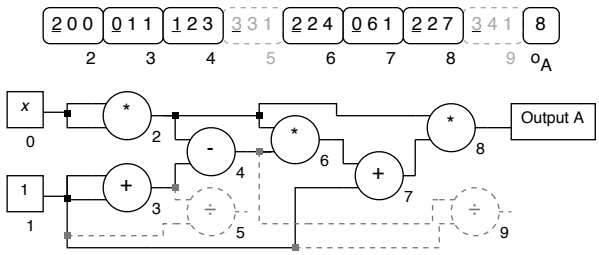

Figure 1: A CGP genotype and corresponding phenotype for the function $x^6 - 2x^4 + x^2$. The underlined genes in the genotype encode the function of each node, the remaining genes encode the node inputs. The function lookup table is: +(0), -(1), *(2), ÷(3). The index labels are shown underneath each program input and node. The inactive areas of the genotype and phenotype are shown in grey dashes.

3. ATTEMPTS AT CROSSOVER IN CGP

This section of the paper reports on some attempts at crossover when the CGP representation is in its original form (as described in the previous section). Four variations of crossover have been tested, but all four failed to improve the convergence of the CGP. Compared to running the CGP with mutation only, the addition of these crossover techniques actually hindered its performance. It is for this reason most people use the CGP without crossover (i.e. using mutation only). Results of two of the four crossover techniques are given here and these results emphasise the need for the new representation and crossover introduced later in the paper.

The crossover methods have been tested on a very simple regression problem given by the equation $x^2 + 2x + 1$. A sample of twenty data points are taken from the interval [0,1], and the cost function is defined as the sum of the squared differences between the population member's values and the true function values at each of the data points. A population size of 30 has been used with 28 offspring created at each generation. Tournament selection has been chosen to select the parents and a mutation rate of 20% has been used. The maximum number of nodes has been set at 5. For each crossover technique the CGP has been run 1000 times and the average convergence over these 1000 runs is recorded at each generation.

The first crossover technique is based on the single point crossover in a binary GA; we treat the nodes in the CGP representation the same as the binary digits in the binary GA. A random node is chosen in the CGP genotype and the offspring are created by swapping the parents nodes at this point (i.e. the first offspring will take all nodes from

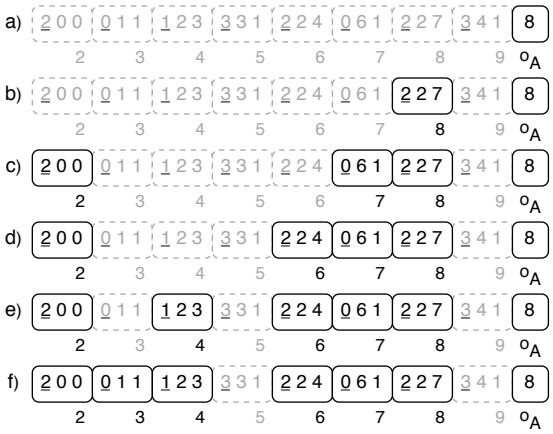

Figure 2: The decoding procedure of a CGP genotype for the function $x^6 - 2x^4 + x^2$. a) Output A (o_A) connects to the output of node 8, move to node 8. b) Node 8 connects to the output of nodes 2 and 7, move to nodes 2 and 7. c) Nodes 2 and 7 connect to the output of node 6 and program inputs 0 and 1, move to node 6. d) Node 6 connects to the output of nodes 2 and 4, move to node 4, as node 2 has already been decoded. e) Nodes 4 connects to the output of nodes 2 and 3, move to node 3. f) Node 3 connects to program input 1. When the recursive process has finished, the genotype is fully decoded.

parent one to the left of this node and all nodes from parent two to the right of this node). Figure 3 displays the average convergence for the two cases; (a) mutation only with a rate of 20% (b) 50% crossover with mutation at 20%. It can be seen that the addition of crossover slows the convergence of the CGP rather than improving it.

The second crossover technique involves picking a random node in the CGP genotype and the offspring are created by swapping this single node in the parents. Figure 4 displays the detrimental effect of this crossover technique. Two other crossover techniques were tried with similar results to those in Figures 3 and 4. It seems that swapping the integers (in whatever manner) in the CGP representation disrupts the performance of the CGP. This has been the motivation for the introduction of the real-valued representation and new crossover technique described in this paper.

4. INTRODUCING THE NEW METHOD

The proposed crossover method for CGP is heavily inspired by the real-valued crossover operator found in real-valued GAs. Normally the CGP genotype consists of a list of integers to encode the directed graph (as described in Section 2). However, to incorporate this type of crossover operator into CGP requires a modification to the CGP representation itself. The modified representation introduces a new level of encoding into the CGP genotype, which represents the directed graph as a fixed length list of real-valued numbers. Each real-valued number corresponds to a single gene in the CGP genotype (as is the case with the standard CGP representation) and its value lies in the range $[0, 1]$. Each node in CGP is still represented by a number of genes and the purpose of each gene still remains as it would

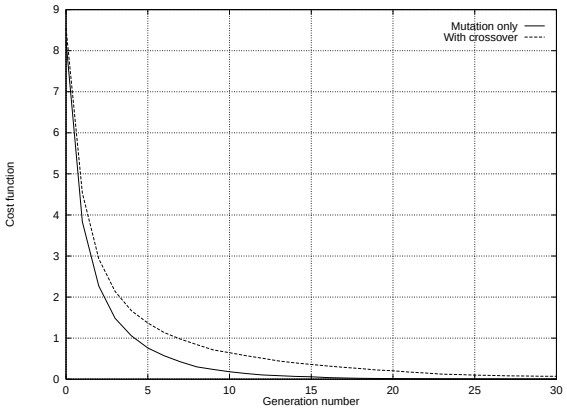

Figure 3: Average convergence of CGP with and without the first crossover technique

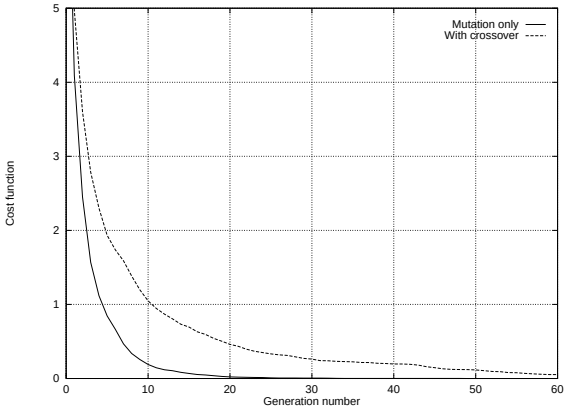

Figure 4: Average convergence of CGP with and without the second crossover technique

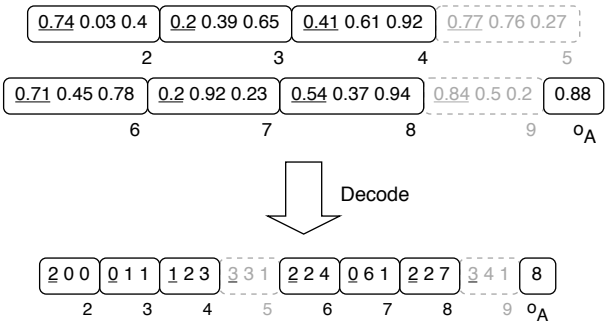

Figure 5: The decoding process between the real-valued and integer-based genotypes. The underlined genes encode the functions and the remaining genes encode the node inputs.

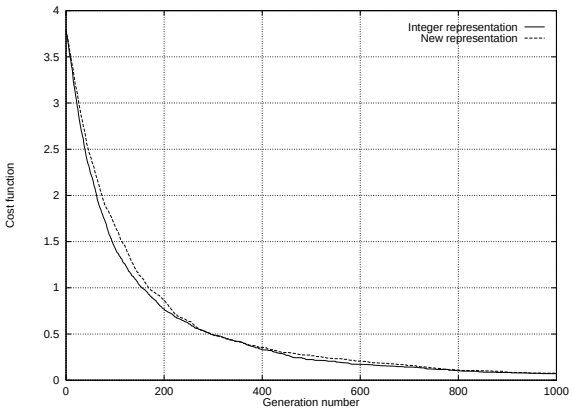

Figure 6: Comparison of the integer CGP with the real-valued CGP without crossover

in the standard CGP genotype; the first real valued gene encodes the function of the node whilst the remaining real-valued genes encode the inputs of the node. An example of the new representation is shown in Figure 5, which also shows the decoding process to the standard CGP genotype.

The decoding process from the real-valued genotype to the integer-based genotype is achieved by the following method. For genes which encode a function (and supposing the function lookup table is +(0), -(1), *(2), ÷(3)) then the interval [0,1] is split into four equal segments. If the real value lies in the first of these segments (i.e. the real value lies between 0 and 0.25) then the real value represents the integer function +(0), if the real value lies in the second segment (i.e. the real value lies between 0.25 and 0.5) it represents the integer function -(1). Similarly a value between 0.5 and 0.75 represents the function *(2) and a value between 0.75 and 1 represents the function ÷(3).

For genes which encode a node input, the interval is split into a number of equal sized segments depending on the number of possible node inputs at that point in the string of numbers. If there were k possible inputs then the interval is split into k segments. If the real value lies between 0 and $1/k$ then it represents integer input 0, if the real value lies between $1/k$ and $2/k$ it represents the integer input 1 and so forth.

By introducing the new representation, each individual in the population can be thought of as a particular value of a function of n variables, where n is the number of real values in the string of numbers.

$$f(x_1,\ x_2, \ldots,\ x_n) \tag{1}$$

The optimization then becomes that of finding the values of these n variables which produce an optimal result.

Crossover is performed as in a floating point Genetic Algorithm. Two parents, p_1 and p_2 are chosen and crossover is performed using Equation 2 to produce two offspring, o_1 and o_2. A uniformly generated random number, r_i, is chosen for each offspring, o_i where $0 < r_i < 1$ and $0 <= i < 2$.

$$o_i = (1 - r_i) * p_1 + r_i * p_2 \tag{2}$$

The mutation operator for the real-valued representation is based on the mutation operator normally found in CGP, the only difference is that it changes the value of a gene to a uniformly generated random real-valued number from the region $[0, 1]$.

Without crossover, the new real-valued representation does not change the behaviour of the CGP very much at all. This can be seen in Figure 6 which displays the average convergence of the CGP over 1000 runs using mutation only for the two CGP representations; the original integer representation and the new real-valued representation. This test has been performed on the first regression problem described in the next section on experimental results.

Using the new crossover method described in this section means that, mathematically, the problem has become that of simply optimising a function of real-valued variables. Instead of randomly changing the input to some complex composite function (as in the case of tree crossover) to attempt to achieve a better solution, the values of the genes are free to slide continuously around the problem space searching for the best solution.

5. EXPERIMENT DETAILS

The new method has been tested on two of the regression problems investigated by Koza, and their equations are given in Equations 3 and 4 below.

$$x^6 - 2x^4 + x^2 \tag{3}$$

$$x^5 - 2x^3 + x \tag{4}$$

A sample of fifty data points are taken from the interval [-1,1], and the cost function is defined as the sum of the absolute values of the differences between the population member's values and the true function values at each of the data points. The algorithm is classed as converged when all of these absolute values are less than 0.01 (this is the criteria Koza used for convergence).

A population size of 50 has been used with 48 offspring created at each generation. Tournament selection has been chosen to select the parents and crossover as described by Equation 2 has been used. The maximum number of nodes has been set at 10 initially and a mutation rate of 20% has been used. Different rates of crossover have been investigated, 0%, 25%, 50% and 75%. Note that 0% crossover is equivalent to the traditional CGP which uses mutation only, although the traditional CGP has been applied with

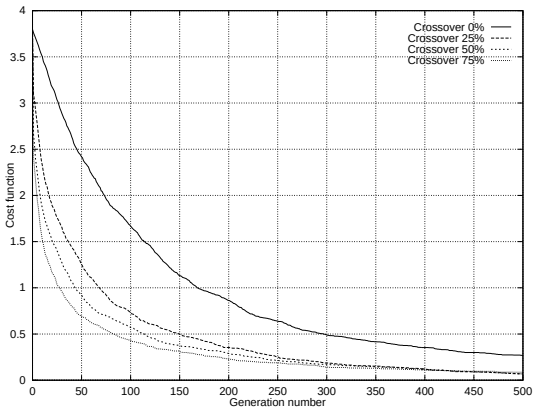

Figure 7: Average convergence for CGP with various crossover rates on $x^6 - 2x^4 + x^2$

a smaller mutation rate and population size in most work prior to this. For each crossover rate the new algorithm has been run 1000 times and the average convergence over these 1000 runs is recorded at each generation.

6. RESULTS

For all the figures in this section of the paper, the horizontal axis represents generation number in the CGP and along the vertical axis is the cost function for the best member of the population (averaged over 1000 runs) for that particular generation number. Figure 7 displays this average convergence (over the 1000 runs of the CGP) for each crossover rate for the regression problem in Equation 3.

From Figure 7, it is apparent that this new form of crossover has a large effect on convergence (unlike tree crossover). If Figure 7 is displayed for the latter generations (see Figure 8), then it can be seen that although the new crossover improves convergence for the initial generations, it does not particularly improve convergence for the latter generations. It is not clear at this stage why this should be the case, but future work will involve investigating possible reasons why. For now, we accept that crossover works better for the initial generations and try a crossover technique which varies with generation number. Since for the initial generations it seems that the larger the crossover rate the faster the convergence, we choose an initial crossover rate for generation number one of 90%. Also since it seems that by generation number 200, crossover is not having a large effect on convergence, we arrange that crossover is 0% by this generation. Therefore for our variable crossover we begin at generation number one with 90% crossover and reduce the crossover rate linearly such that by generation number 180 crossover is being performed 0% of the time. This variable crossover technique is simply based on analysing these initial results, future work will involve investigating alternative variable crossover techniques. Figure 9 displays the average convergence for the variable crossover technique and it can be seen that this means a faster convergence over all generation numbers.

Table 1 displays the average number of generations required to reach convergence and the computational effort as described by Koza in [6] and shown in Equation 5.

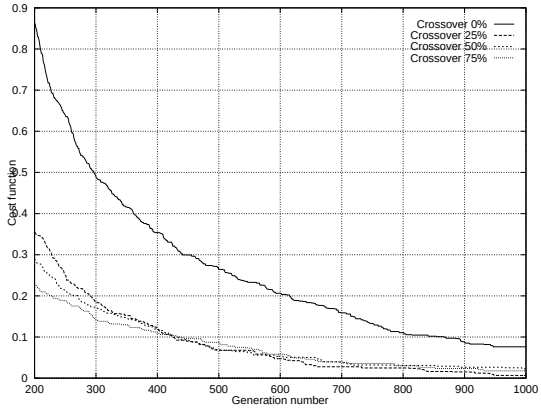

Figure 8: Average convergence of CGP for the latter generations on $x^6 - 2x^4 + x^2$

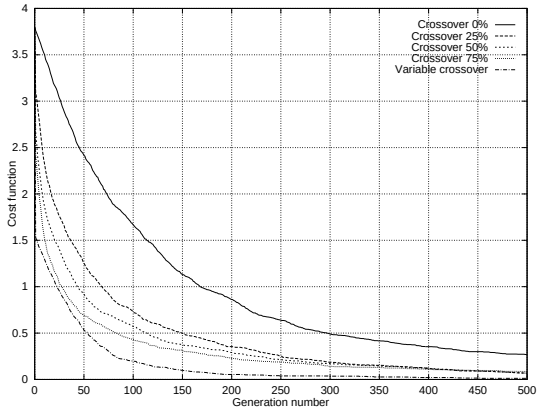

Figure 9: Average convergence for CGP with various crossover rates, including a variable crossover rate on $x^6 - 2x^4 + x^2$

Table 1: The average number of generations and computational effort (CE) required by CGP with ten nodes to converge on a solution for $x^6 - 2x^4 + x^2$

Crossover Rate (%)	Average Generations	CE	U
0	168	30,000	-
25	84	9,000	309,778 ‡
50	57	8,000	261,533 ‡
75	71	6,000	226,303 ‡
Variable	47	10,000	263,269 ‡

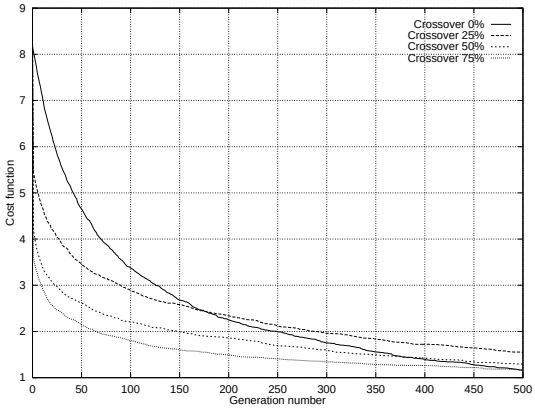

Figure 10: Average convergence for the second regression problem $x^5 - 2x^3 + x$

The significance of the results is also assessed using the non-parametric Mann-Whitney U test [10]. The U values produced from the Mann-Whitney U test are denoted with: a * if they are classed as marginally significant ($P < 0.05$), a † if they are classed as significant ($P < 0.01$) or a ‡ if they are classed as highly significant ($P < 0.001$).

$$\begin{aligned} P(M,i) =& \frac{N_s(i)}{N_{total}} \\ R(z) =& ceil\left(\frac{\log(1-z)}{\log(1-P(M,i))}\right) \\ min\, I(M,i,z) =& MR(z)\, i + 1 \end{aligned} \tag{5}$$

Figure 10 displays the average convergence for the regression problem given in Equation 4. Note that for this problem it is more pronounced that the crossover has a big effect on convergence for the initial generations but has less effect for the latter generation. Variable crossover improves the convergence over all generations, as can be seen in Figure 11.

Table 2 contains the average number of generations required to converge together with Koza's computational effort figure for the various percentages of crossover.

For this second regression problem, crossover does not seem to have as big an effect as for the previous problem. This seems to be because, for this problem, occasional runs take a huge number of generations to converge. This can be seen in Figure 12 which shows the number of generations required to converge for 100 runs of the two regression problems. As can be seen in the figure, for the first problem most runs take approximately the same number

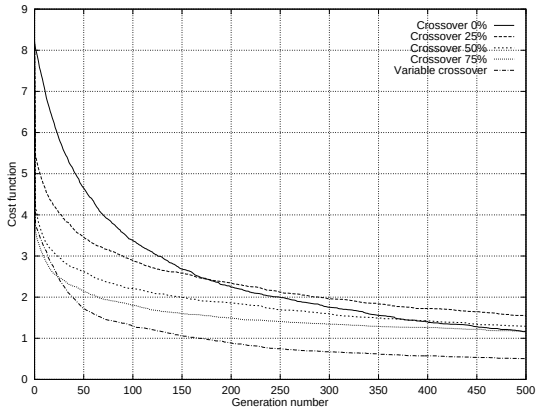

Figure 11: Average convergence for the second regression problem $x^5 - 2x^3 + x$ including results for a variable crossover rate

Table 2: The average number of generations and computational effort (CE) required by CGP with ten nodes to converge on a solution for $x^5 - 2x^3 + x$

Crossover Rate (%)	Average Generations	CE	U
0	516	44,000	-
25	735	24,000	502,024
50	691	14,000	422,394 ‡
75	655	11,000	343,119 ‡
Variable	278	13,000	294,577 ‡

of generations to converge, whereas in the second problem there are occasional runs which take a very large number of generations to converge. This trait will be investigated in more detail in future work.

The number of nodes used is now increased from 10 to 50 in both regression problems. Figure 13 displays the results for the regression problem in Equation 3, and Table 3 gives the average number of generations to converge together with Koza's computational effort figure. Figure 14 and Table 4 are the same for the regression problem given in Equation 4.

The results in this section show that the new technique enhances the performance of CGP. The majority of the U values produced are classed as highly significant, which supports the findings from computational effort figures and indicates that the use of crossover in CGP is beneficial when

Table 3: The average number of generations and computational effort (CE) required by CGP with fifty nodes to converge on a solution for $x^6 - 2x^4 + x^2$

Crossover Rate (%)	Average Generations	CE	U
0	78	18,000	-
25	85	13,000	443,769 ‡
50	71	11,000	420,519 ‡
75	104	13,000	463,118 †
Variable	45	14,000	401,205 ‡

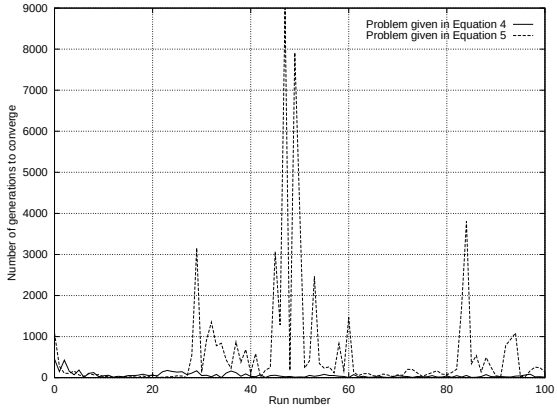

Figure 12: The number of generations to converge over 100 runs for both symbolic regression problems

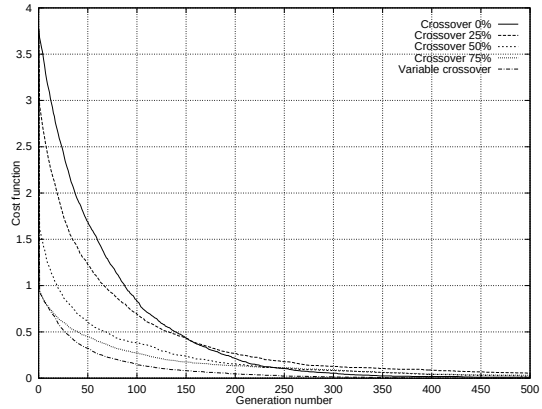

Figure 13: Average convergence for the symbolic regression problem in Equation 3 using CGP with fifty nodes

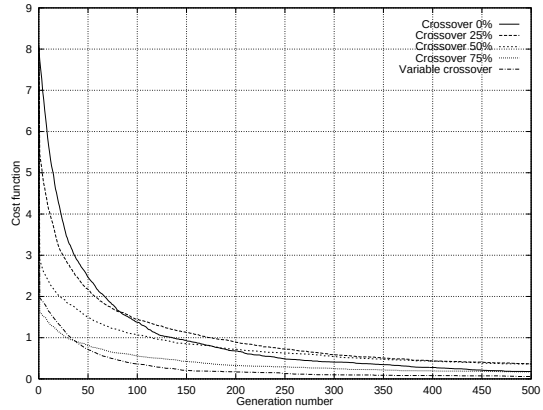

Figure 14: Average convergence for the symbolic regression problem in Equation 4 using CGP with fifty nodes

Table 4: The average number of generations and computational effort (CE) required by CGP with fifty nodes to converge on a solution for $x^5 - 2x^3 + x$

Crossover Rate (%)	Average Generations	CE	U
0	131	18,000	-
25	193	17,000	539,076 †
50	224	12,000	454,875 ‡
75	152	19,000	554,642 ‡
Variable	58	16,000	470,984 ⋆

applied to symbolic regression problems. The reason the new method works well could be the fact that the problem has been transformed into that of simply minimising a function (the cost function) of n variables (where n is the total number of real-valued numbers in the representation). Crossover methods tested in the past have involved swapping the integers in the CGP representation in some manner, and it is thought that this may produce too great a change to the functional form of the current solution. By making the cost function into a simple function of variables and performing crossover in the way described in this paper, the values of the variables are allowed to move in a continuous manner to their optimal values.

It is also thought that another possible reason for the success in the new technique may be attributed to the fact that for nodes to the far left of the representation, the interval [0,1] is spit into a less number of sub-sections and therefore it will "change" less due to the crossover. In contrast for nodes to the far right of the representation, the interval [0,1] is split into more sub-sections and therefore is more likely to change through crossover. It is thought that this could help the optimisation due to the fact that functions to the left can be thought of as fundamental sub-functions of the entire solution function.

7. CONCLUSIONS AND FUTURE WORK

In this paper we have introduced a new crossover technique, which improves the performance of Cartesian Genetic Programming. The CGP representation is adapted slightly in order to allow the new crossover. It has been found that this new representation together with crossover reduces the average number of generations required to converge by 72% in the case of the regression problem given in Equation 3, and by 46% in the case of the problem in Equation 4. It has been shown that for the regression problem in Equation 4 the new crossover technique does not have as good an effect on convergence as for the first regression problem. This is thought to be because of the fact that occasional runs of the CGP for this second problem take a huge number of generations to converge. Future work will involve investigating this trait.

The results in this paper for the cases where crossover is set at 0% are equivalent to those of the traditional CGP, which uses mutation only. The computational effort figures reported in this paper for 0% crossover are similar to those reported for the traditional CGP [18], although in this paper a larger mutation rate and population size have been used. Future work will involve investigating how changing these parameter values in the CGP (i.e. mutation rate, population

size, parent selection method) affects the performance of the new method. We will also investigate the fact that crossover has more effect for the initial generations and try alternative method of variable crossover.

This paper reports on initial testing of the new technique when applied to two regression problems. Future work will involve testing the new method on other problems, in particular on larger problems and other types of problems.

8. REFERENCES

[1] P. Angeline. Subtree crossover: Building block engine or macromutation? In *Genetic Programming 1997: Proceedings of the Second Annual Conference (GP97)*, pages 9–17, Stanford University, USA, 13–16July 1997. Morgan Kaufman.

[2] J. Clegg, J. Dawson, S. Porter, and M. Barley. The use of a genetic algorithm to optimize the functional form of a multi-dimensional polynomial fit to experimental data. In *2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 928–934, Edinburgh, 2005.

[3] J. Clegg, A. Marvin, J. Dawson, and S. Porter. Optimisation of stirrer designs in a reverberation chamber. In *IEEE Trans. EMC*, volume 47 of *No. 2*, pages 399–403, 2005.

[4] L. Dawson, J. Clegg, S. Porter, J. Dawson, and M. Alexander. The use of genetic algorithms to maximise the performance of a partially lined screened room. In *IEEE Trans. EMC*, volume 44 of *No. 1*, pages 233–242, 2002.

[5] K. De Jong. An analysis of the behaviour of a class of genetic adaptive systems. In *Doctoral Thesis, Department of Computer and Communication Sciences. University of Michigan, Ann Arbor.*, 1975.

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

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

[8] S. Luke and L. Spector. A comparison of crossover and mutation in genetic programming. In *Genetic Programming 1997: Proceedings of the Second Annual Conference (GP97)*, pages 240–248, Stanford University, USA, 13–16 July 1997. Morgan Kaufman.

[9] S. Luke and L. Spector. A revised comparison of crossover and mutation in genetic programming. In *Genetic Programming 1998: Proceedings of the Third Annual Conference (GP98)*, pages 208–213, University of Wisconsin, Madison, WI, USA, 22–25 July 1998. Morgan Kaufman.

[10] H. Mann and D. Whitney. On a test of whether one of 2 random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, (18):50–60, 1947.

[11] J. F. Miller. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *GECCO 1999: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1135–1142, Orlando, Florida, 1999. Morgan Kaufmann.

[12] J. F. Miller and P. Thomson. Cartesian genetic programming. In *Proceedings of the 3rd European Conference on Genetic Programming (EuroGP 2000)*, volume 1802 of *Lecture Notes in Computer Science*, pages 121–132, Edinburgh, 2000. Springer-Verlag.

[13] R. Poli. Parallel Distributed Genetic Programming. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 403–432. McGraw-Hill, UK, 1999.

[14] J. Schaffer and L. Eshelman. On crossover as an evolutionarily viable strategy. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 61–68, La Jolla, CA, 1991. Morgan Kaufmann.

[15] W. Spears and K. De Jong. On the virtues of uniform crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236, La Jolla, CA, 1991. Morgan Kaufmann.

[16] V. K. Vassilev and J. F. Miller. The advantages of landscape neutrality in digital circuit evolution. In *Proceedings of the 3rd International Conference on Evolvable Systems (ICES 2000)*, volume 1801 of *Lecture Notes in Computer Science*, pages 252–263. Springer Verlag, 2000.

[17] M. Vose and G. Liepins. Schema disruption. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 237–242, La Jolla, CA, 1991. Morgan Kaufmann.

[18] J. Walker and J. Miller. Automatic acquisition, evolution and re-use of modules in cartesian genetic programming. *to be published in IEEE Transactions on Evolutionary Computation*, 2007.

[19] T. Yu and J. F. Miller. Neutrality and the evolvability of boolean function landscape. In *Proceedings of the 4th European Conference on Genetic Programming (EuroGP 2001)*, volume 2038 of *Lecture Notes in Computer Science*, pages 204–217. Springer-Verlag, 2001.