International Journal of Automation and Computing 05(1), January 2008, 81-89
DOI: 10.1007/s11633-008-0081-4

Efficient Graph-based Genetic Programming Representation with Multiple Outputs

Edgar Galvan-Lopez
Department of Computer Science, University of Essex, Colchester CO4 3SQ, UK

Abstract: In this work, we explore and study the implication of having more than one output on a genetic programming (GP) graph-representation. This approach, called multiple interactive outputs in a single tree (MIOST), is based on two ideas. First, we defined an approach, called interactivity within an individual (IWI), which is based on a graph-GP representation. Second, we add to the individuals created with the IWI approach multiple outputs in their structures and as a result of this, we have MIOST. As a first step, we analyze the effects of IWI by using only mutations and analyze its implications (i.e., presence of neutrality). Then, we continue testing the effectiveness of IWI by allowing mutations and the standard GP crossover in the evolutionary process. Finally, we tested the effectiveness of MIOST by using mutations and crossover and conducted extensive empirical results on different evolvable problems of different complexity taken from the literature. The results reported in this paper indicate that the proposed approach has a better overall performance in terms of consistency reaching feasible solutions.

Keywords: Interactivity within an individual (IWI), multiple interactive outputs in a single tree (MIOST), neutrality, evolvable hardware, genetic programming (GP).

1 Introduction

Genetic programming (GP)[1] is a heuristic search technique that has its inspiration from the theories of genetic inheritance and natural selection. This technique has been proved to be a suitable tool for solving problems in many applications. Usually, in GP, programs are expressed as syntax trees. Despite the vast number of good results reported when using GP[1], there are some researchers that have proposed different types of representations or additions to the traditional form of GP, i.e., tree-like structures.

For example, Koza[2] proposed automatically defined functions (ADFs). ADF is a function (subprogram, procedure or module) that is dynamically evolved during a run of a GP. Typically, the ADFs process one or more dummy arguments. The problem with this approach is discovering good ADFs. ADFs behave differently in different parts of a program when they have different arguments. Thus, to discover if an ADF is good, GP has to spend computation time to discover with which parameters the ADF can be used properly.

Angeline and Pollack[3] proposed a method called evolutionary module acquisition (EMA). The idea of this method is to build and evolves modules (which are the reuse of code) during the evolution process. To identify appropriate module(s) in the evolving individuals, the authors add two operators to the reproduction process. The first operator, called compress, selects a portion of the offspring to preserve for future manipulation. The second operator, called expand, is the opposite to the former operator. Because there is no general method of identifying what portions of the individual should be compressed, the composition of each module is selected randomly. The same authors extended this work in [4]. In this work, the authors referred to the method as genetic library builder (GLiB). This library is a collection of all created modules and serves only as reference symbol for GLiB. During either a genotype′s execution or the expansion of the compressed module, GLiB retrieves the definitions of symbols from the genetic library.

Montana[5] proposed a variation of GP called strongly typed genetic programming (STGP). He started from the definition of closure (which means that all elements, functions and terminals, take arguments of a single data type and return values of the same data type). Koza[1] described a way to relax this constraint of closure with the concept of constrained syntactic structures. He used tree generation routines which only generate legal trees, and used operator which maintains legal syntactic structures. However, the main characteristic of STGP is to build an individual as a parse tree and the data type of the nodes not necessarily should be the same type.

Teller and Veloso[6] were one of the first researchers to use a graph-based GP. Their method, parallel algorithm discovery and orchestration (PADO), is a combination of GP and linear discriminator which was used to obtain parallel classification programs for signals and images.

Poli[7] proposed an approach called parallel distributed genetic programming (PDGP). Poli stated that PDGP can be considered as a generalization of GP. However, PDGP has more complex representations and evolves finite state automata, neural networks and more. PDGP is based on a graph-like representation for parallel programs which is manipulated by crossover and mutation operators and guarantee the syntactic correctness of the offsprings. Poli′s approach was inspired by the parallel distributing processing performed in neural networks.

Manuscript received September 17, 2007; revised November 20, 2007
This paper was supported by the Mexican Consejo Nacional de Ciencia y Tecnologia (CONACyT) for the postgraduate studies at University of Essex.
E-mail address: egalva@essex.ac.uk

Angeline[8] proposed a representation called multiple interacting programs (MIPs). This representation is a generalization of a recurrent neural network that can model any type of dynamic system. Each program in a given set is unique and stored in the form of a parse tree. Using this technique, an individual is virtually equivalent to a neural network where the computation performed at each unit is replaced with an independent evolved equation.

Miller and Thomson[9] proposed the Cartesian genetic programming (CGP). This technique was called Cartesian in the sense that the method considers a grid of nodes that are addressed in a Cartesian coordinate system. In CGP, the genotype is represented as a list of integers that are mapped to directed graphs rather than trees. CGP had its original motivation from the effectiveness of the approach in learning Boolean functions[10].

Kantschik and Banzhaf[11] proposed a different representation of GP named linear-tree. The main idea was to give flexibility to a program to choose different execution paths for different inputs. In this method, each program is represented as a tree. Each node in the tree has two parts, a linear program and a branching node. The linear program can be executed when the node is reached during the interpretation of the program. The branching node indicates the possible program flow. The authors claimed that their representations have better performance than with the tree-based representation. Later on, the same authors proposed a representation called linear-graph[12]. They argued that graphs come one step nearer to the control flow of a hand-written program. In their approach, they have shown that this method is better than the linear-tree representation.

As can be seen from the previous summaries, many and diverse ideas have been raised in evolutionary computation systems (ECs), specifically in the paradigm of GP to make it more powerful and efficient.

The main purpose of the present work is to present a new GP technique, called multiple interactive outputs in a single tree (MIOST), which allows evolving graph-like structures and it allows to have more than one output in an individual structure. To study this form of GP, we have sub-divided our study in two different steps:

- First, we explore the idea of a graph-GP representation by introducing pointers in the individuals.
- Second, we add to the above representation, as many outputs as the user requires.

In this paper, we will use evolvable hardware problems to test MIOST.

2 Approach

Our approach, called MIOST, is the result of two main ideas:

1) Using a rich graph-GP representation by allowing pointers in individuals.

2) Defining in an individual as many outputs as the user requires.

To verify the efficiency of our approach, we study it in two stages. That is, we first study the impact of having pointers within the individuals, which we call this interactivity within an individual (IWI). Once we have studied the implications of allowing pointers in the individuals (IWI), we add multiple outputs to the individuals (MIOST).

2.1 Interactivity within an individual (IWI)

Terminal and function set

The representation used in our work is a tree-like structure as suggested in [1]. The terminal set consists of the inputs of the circuit (i.e., $T = \{a, b, c, \cdots\}$). The function set is composed by the typical operators used in these types of problems (i.e., AND, OR, NOT, etc.). The function set includes also a special function p. By using the p symbol, we have taken inspiration from graph-GP representations. We decided to use and explore the possibility of representing programs as graphs with oriented links. Fig. 1 depicts this idea.

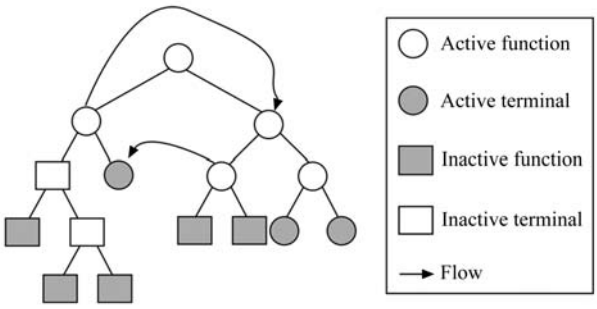

Fig. 1 A typical individual created using IWI

The idea was to replace a function node by an element that represents links that determines what needs to be evaluated. We hope in this way to find parts of an individual that can be more useful in other part(s) of the same individual. Note that the role of the function node is very different from ADFs[2] where a function can be a subprogram, procedure or module. The main difference, is that in IWI a function node just refers to a place where that function will follow the execution.

Let us discuss more in detail how the p symbol works:

- Once the individuals in the population have been generated, we use a probability to replace any function with a p symbol which is a function of arity 2 [1].
- If an individual contains in its structure a p symbol, this will point to code somewhere in the program so, when p is executed, the subtree rooted at that node is ignored.
- If the p symbol points to a function symbol, the p symbol effectively represents the subtree rooted at that function.
- If the p symbol points to a terminal symbol, the p symbol simply represents that node.

[1] This is not a restriction because p could be of any arity.

2.2 Multiple interactive outputs in a single tree (MIOST)

MIOST is an extension of the previous approach (IWI). Fig. 2 depicts an individual created with MIOST. Individuals created with MIOST could have in their structures p symbols and also more than one output. For this purpose, it is necessary to define an extra set apart of the terminal and function set. This is explained in the following paragraphs.

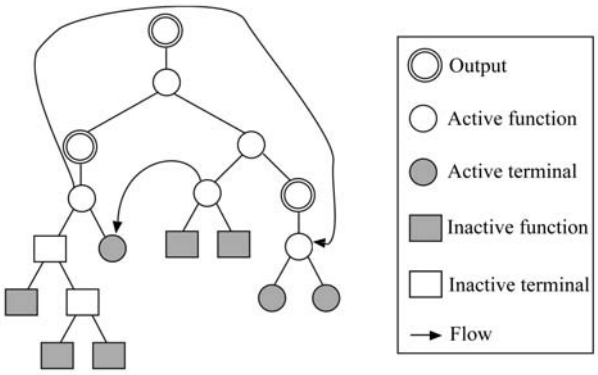

Fig. 2 A typical individual created using MIOST

Output set

When we create an individual using MIOST, it is necessary to define an output set which will contain as many outputs as the user requires. That is, if we need to create individuals of four outputs then the output set will look like $O = \{O_1, O_2, O_3, O_4\}$. Once the output set has been defined, the individual is created using the three sets: terminal, function, and output set.

When an individual is created, the first set that must be used is the output set, the system will choose randomly any element from the output set and the chosen element cannot be selected again for the current individual and so on until the individual is created. Once an individual has been created, we check if it contains all the outputs defined in the output set, if not the process is repeated until a valid individual has been created. It is worth mentioning that when we create an individual we do not specify which output will be in the root, so our approach has the advantage to place the most complex output in the root. The results confirm this.

2.3 Genetic operators

The crossover operator used in IWI and MIOST works as usual but an important difference with the traditional crossover used in GP is that, if the subtree swapped contains a p symbol, the p symbol$'$s pointer is not changed [2]. Moreover, in the case of MIOST, there is another difference: once we have created our individual in the population, we classify each node of each individual to know which nodes can be used to apply crossover. With this, we assure that an individual will contain the number of outputs that it must contain.

The mutation operator is applied as usual on a per node basis. The only restriction is that a p symbol is not allowed to be mutated.

[2]There is an exception to this rule: we prevent a p symbol from referring to a subtree that contains the same p since this would lead to an infinite loop. We do this by reassigning the position to where p in question is pointing to.

2.4 Fitness function

To test the effectiveness of IWI and MIOST, we have used several evolvable hardware problems of different complexity taken from the literature. The fitness function works in two stages:

1) At the beginning of the search, the fitness of a genotype is the number of correct output bits (raw fitness).

2) Once the fitness has reached the maximum number of correct output bits, we try to optimize the circuits by giving a higher fitness to an individual with shorter encodings.

2.5 Features

The previous approaches have interesting features. For instance, the presence of p symbols in the representations, assure us that there are inactive code in the individuals. This has at least two advantages:

1) When a mutation takes place in inactive code, there is no need to evaluate an individual since there is a change at genotype level but not at phenotype level;

2) It allows to study neutrality[13] which is considered an area of controversial debate on EC systems. There are, however, some works that have shed some light on neutrality[14−17].

3 Effects of graph-GP representation

Before testing the approach explained earlier (MIOST), we will conduct simple experiments (mutation-based, tree-like GP system without crossover) using different p rates and mutation rates. We used a well-known benchmark problem (6-bit multiplexer Boolean function) to conduct our analysis.

To obtain meaningful results, we performed 20 independent runs for each of the different mutation and p rates used in our studies. We have used 200 individuals, $depth = 8$, and 400 generations. For these experiments, we have used the full initialization method to create the individuals in the population. Crossover operator was not used.

From Fig. 3, we can see the success rates using the graph-GP representation using different mutation and p rates. The highest success rate found was 100% when set $p = 0.01$ and mutation rate is equal to 0.02. Keeping the value of p constant and increasing the mutation rates, the success rate tends to decrease. Similar behaviour can be observed with different p rates. We can conclude that regardless of the value of p, the higher the mutation rate is, the lower the success rate will be.

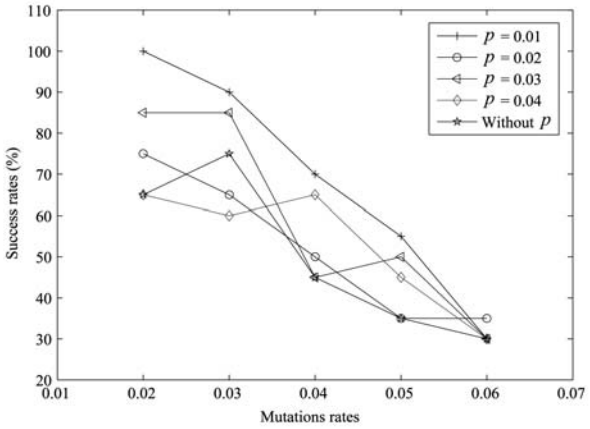

Fig. 3 Success rates obtained using different mutation and p rates on the 6-bit multiplexer Boolean function

Table 1 shows the results of using the graph-GP representation with only the mutation operator. p and mutation rates are shown in the first and second column, respectively. Feasible circuits (success rate) and the average number of generations that are necessary to reach the feasible zone are shown in the last two columns.

Table 1 Results of using the graph-GP representation with only the mutation operator

p	Mutation rate	Feasible circuits	Average of generations
0.01	0.02	100%	80.6
0.01	0.03	90%	155.66
0.01	0.04	70%	138.62
0.01	0.05	55%	146.65
0.01	0.06	30%	203.5
0.02	0.02	75%	91.46
0.02	0.03	65%	136.66
0.02	0.04	50%	141.5
0.02	0.05	35%	163.71
0.02	0.06	35%	243.14
0.03	0.02	85%	122.7
0.03	0.03	85%	109.64
0.03	0.04	45%	179.5
0.03	0.05	50%	237.5
0.03	0.06	30%	157.56
0.04	0.02	65%	109.65
0.04	0.03	60%	117
0.04	0.04	65%	132.45
0.04	0.05	45%	220.54
0.04	0.06	30%	164

At this point, one question arises: what happens if we do not allow the presence of the p element in our individuals? To answer this question, we need to take a look at Fig. 3. In no case was the system able to reach a success rate of 100% in the absence of the p symbol. Moreover, the performance of the GP system without the presence of p is poor compared to when it is present. In fact, the performance of the GP system when p is not present in the individuals is the worst for all mutation rates, except when mutation rate is 0.03.

As mentioned previously, one of the features of the proposed approach is the presence of neutrality proposed by Kimura[13]. Kimura[13] put forward the theory that the majority of evolutionary changes at the molecular level are the result of random fixation of selectively neutral mutations. In other words, the mutations that take place in the evolutionary process are neither advantageous nor disadvantageous to the survival of individuals. Kimura′s theory considers a mutation from one gene to another as neutral if this modification does not affect the phenotype.

The results presented previously show how the individuals in the population tend to behave in the presence of p in their structures. Fig. 3 summarizes this behaviour on four different p rates. The highest success rates were found when the mutation rates were set with the lowest value (0.02), regardless of the p rates. At the beginning of the evolutionary process, the number of individuals affected by neutral mutations is high but it tends to decrease after a few generations (see Figs. 4–7). In other words, individuals with p elements in their structures tend to disappear at the beginning of the process. We think this happens because at the beginning of the evolutionary process the solution needs to be protected by allowing the presence of p in their structures.

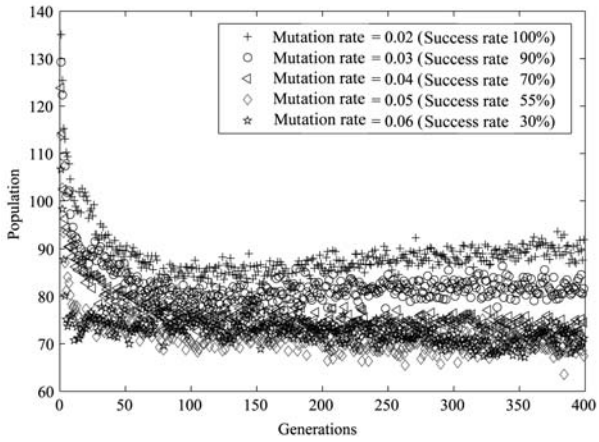

Fig. 4 Individuals affected by neutrality ($p = 0.01$)

Around 50-60 generations, when the number of individuals is affected by neutral mutations, these became stable. As can be observed in Figs. 4–7, the best performance is achieved when the number of individuals affected by neutral mutation is in the range of 90–100. Notice that this range is close to half of the size of the population. On the other hand, the worst performance was found when the number of individuals affected by neutral mutation is below 80.

From this analysis, it is clear that the presence of p in the individuals can make the solution avoid getting stuck in local optima. However, a fine balance between p rate and mutation rate is needed to improve the exploration of the

search space.

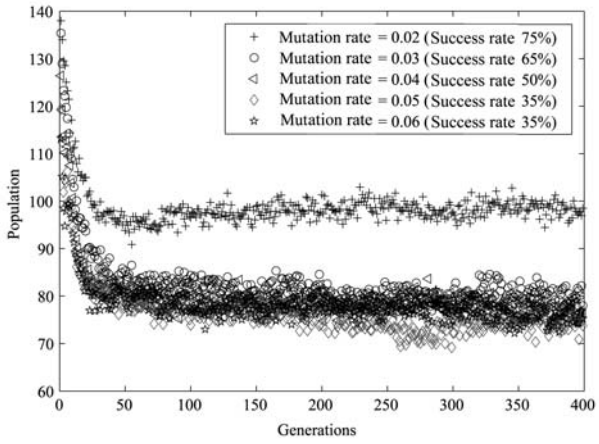

Fig. 5 Individuals affected by neutrality ($p = 0.02$)

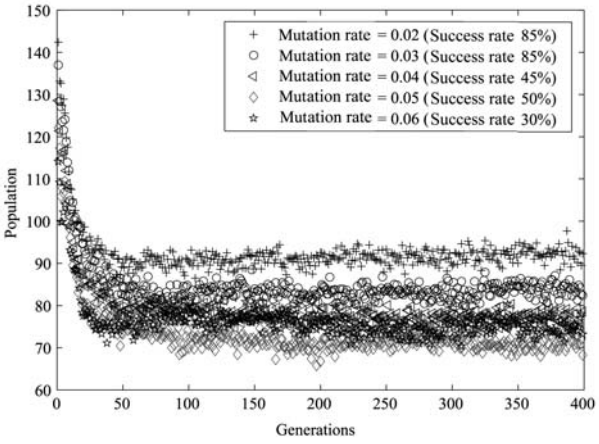

Fig. 6 Individuals affected by neutrality ($p = 0.03$)

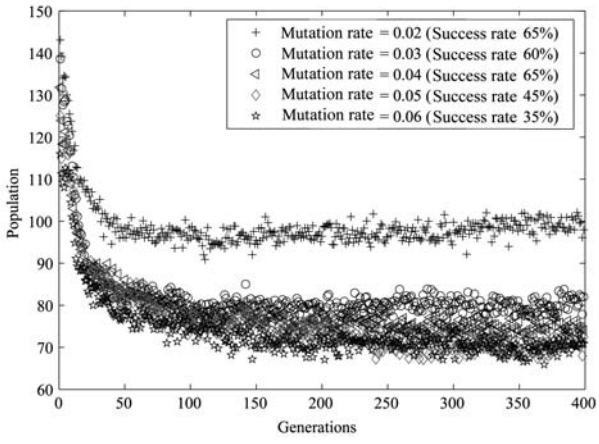

Fig. 7 Individuals affected by neutrality ($p = 0.04$)

Using this example and carrying out experiments using only mutations, we can see that the best performance is achieved when setting $p = 0.01$ and mutation rate is 0.02, and these are the values that we have used to conduct our experiments in more complex problems. Moreover, the previous analysis has helped us to get a good indicator that the graph-GP representation performs well.

4 Comparison of results

We used several evolvable hardware problems of different complexity taken from the literature to test IWI and MIOST. Our results were compared with those obtained by multiobjective genetic algorithm (MGA)[18], a binary single-objective using integer A encoding particle swarm optimization (EAPSO)[19], a binary single-objective using integer B encoding PSO (EBPSO)[20], a binary single-objective PSO (BPSO)[20], encapsulated genetic programming (EGP)[21] and the traditional GP. For all the examples, we performed 20 independent runs. As we will see in the next paragraphs, in all the experiments we improve the percentage of feasible region [3] compared with the other techniques.

4.1 Results using IWI

After a series of preliminary experiments, we have decided to use a crossover rate of 70, mutation rate of 0.02 and p rate of 0.01.

Example 1. For our first example, we have used the truth table shown in Table 2, where a, b, and c denote the inputs and O is the desired output. The parameters used in this example are the following: population size (PS) is 190 and the maximum number of generations (MNG) is 525, i.e., a total of 99 750 fitness function evaluations. The same values parameters were used by GP. BPSO, EAPSO and EBPSO performed 100 000 fitness function evaluations, while MGA performed 102 000. As we can see in Table 3, the algorithms able to converge to a feasible region in 100% of the runs were BPSO, EBPSO, and IWI. Moreover, in IWI, the average of generations at which it solved the circuit was 35.61, while in GP the average of generations was 56.05. However, the average number of gates in IWI was 10.4, while the average number of gates in EBPSO was 6.15.

Table 2 Truth table of Example 1

a	b	c	d	O_1
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

[3] The feasible region is the area of the search space containing circuits that match all the outputs of the problem′s truth table.

Table 3 Comparison of results among BPSO, EAPSO, EBPSO, MGA, GP, EGP, and IWI on Example 1

Methods	Feasible Circuits	Average of gates	Average of generations
BPSO	100%	6.75	-
EAPSO	95%	7.3	-
EABPSO	100%	6.15	-
MGA	90%	9.3	-
GP	90%	11.05	56.05
EGP	-	-	-
IWI	100%	10.4	35.61

Example 2. For our second example, we have used the truth table shown in Table 4. The parameters used in this example are the following: PS = 240 and MNG = 415 (i.e., a total of 99 600 fitness function evaluations). The same values parameters were used by GP. BPSO, EAPSO and EBPSO performed 100 000 fitness function evaluations, while MGA performed 102 000. As we can see in Table 5, the algorithms able to converge to a feasible region in 100% of the runs performed were EBPSO and IWI. Moreover, in IWI the average of generations required to solve the circuit was 28.88, while in GP the average of generations is 49.23. However, the average number of gates in EBPSO is slightly smaller (5.9).

Table 4 Truth table of Example 2

a	b	c	d	O_1
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Table 5 Comparison of results among BPSO, EAPSO, EBPSO, MGA, GP, EGP, and IWI on Example 2

Methods	Feasible Circuits	Average of gates	Average of generations
BPSO	85%	10.4	-
EAPSO	90%	8.25	-
EBPSO	100%	5.9	-
MGA	70%	13.7	-
GP	90%	9.22	49.23
EGP	-	-	-
IWI	100%	8.6	28.88

Example 3. For our third example, we have used the truth table shown in Table 6. The parameters used in this example are the following: PS = 550 and MNG = 900 (i.e., a total of 495 000 fitness function evaluations). The same values parameters were used by GP. BPSO, EAPSO and EBPSO performed 500 000 fitness function evaluations, while MGA performed 528 000. As we can see in Table 7, IWI is the algorithm with the highest percentage of feasible solutions reached (70%). Moreover, in IWI the average of generation at which it solved the circuit was 156.57, while in GP it is 791. However, the average number of gates in IWI is 32.28, while the average number of gates in EBPSO was much smaller (12.15).

Table 6 Truth table of Example 3

a	b	c	d	e	O_1
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	0
0	1	0	0	0	0
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	0
0	1	1	0	0	1
0	1	1	0	1	0
0	1	1	1	0	0
0	1	1	1	1	0
1	0	0	0	0	0
1	0	0	0	1	1
1	0	0	1	0	1
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	0	1	0
1	1	0	1	0	0
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	0

Table 7 Comparison of results among BPSO, EAPSO, EBPSO, MGA, GP, EGP, and IWI on Example 3

Methods	Feasible Circuits	Average of gates	Average of generations
BPSO	-	-	-
EAPSO	50%	13.8	-
EBPSO	55%	12.15	-
MGA	25%	21.4	-
GP	5%	90	791
EGP	-	-	-
IWI	70%	32.28	156.57

In the following paragraphs, we will continue towards the same idea: verifying the effectiveness of using a graph-GP representation and allowing more than one output in the individual′s structure. For this purpose, we will use several evolvable hardware problems of more than one output of different complexity taken from the literature.

4.2 Results using MIOST

Example 4. For our fourth example, we have used the truth table shown in Table 8. The parameters used in this example are the following: PS = 380 and MNG = 525 (i.e., a total of 199 500 fitness function evaluations). The same values parameters were used by EGP, IWI. BPSO, EAPSO, and EBPSO performed 200 000 fitness function evaluations, while MGA performed 201 300. As we can see in Table 9, the only algorithms able to converge to the feasible region in 100% of the runs were EBPSO and MIOST.

Table 8 Truth table of Example 4

a	b	c	d	O_1	O_2
0	0	0	0	1	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	0	1
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	1
1	1	1	1	0	1

Table 9 Comparison of results among BPSO, EAPSO, EBPSO, MGA, GP, EGP, and MIOST on Example 4

Methods	Feasible Circuits	Average of gates	Average of generations
BPSO	95%	10.05	-
EAPSO	70%	13.45	-
EBPSO	100%	7.75	-
MGA	75%	13.4	-
EGP	55%	9.7	122.9
MIOST	100%	12.9	109.55

Example 5. For our fifth example, we have used the truth table shown in Table 10. The parameters used in this example are the following: PS = 1 200 and MNG = 832 (i.e., a total of 998 400 fitness function evaluations). The same values parameters were used by EGP. BPSO, EAPSO, and EBPSO performed 1 000 000 fitness function evaluations, while MGA performed 1 101 040. As we can see in Table 11, MIOST is the algorithm which has the highest percentage of feasible solutions reached (75%). Moreover, in MIOST the average of generations at which it solved the circuit was 104.67, while in EGP it was 149.5. Surprisingly, MIOST is one of the algorithms with the lowest average number of gates (11.6).

Table 10 Truth table of Example 5

a	b	c	d	e	O_1	O_2	O_3
0	0	0	0	0	1	1	0
0	0	0	0	1	1	0	0
0	0	0	1	0	1	1	0
0	0	0	1	1	1	0	0
0	0	1	0	0	1	1	1
0	0	1	0	1	1	0	1
0	0	1	1	0	1	1	0
0	0	1	1	1	1	1	0
0	1	0	0	0	1	1	0
0	1	0	0	1	1	0	0
0	1	0	1	0	1	1	0
0	1	0	1	1	1	0	0
0	1	1	0	0	1	1	1
0	1	1	0	1	1	0	1
0	1	1	1	0	1	1	0
0	1	1	1	1	1	1	0
1	0	0	0	0	0	1	0
1	0	0	0	1	0	0	0
1	0	0	1	0	0	1	0
1	0	0	1	1	0	0	0
1	0	1	0	0	0	1	1
1	0	1	0	1	0	0	1
1	0	1	1	0	0	1	0
1	0	1	1	1	0	1	0
1	1	0	0	0	0	1	0
1	1	0	0	1	0	0	0
1	1	0	1	0	0	1	0
1	1	0	1	1	0	0	0
1	1	1	0	0	1	1	1
1	1	1	0	1	1	0	1
1	1	1	1	0	1	1	0
1	1	1	1	1	1	1	0

Table 11 Comparison of results among BPSO, EAPSO, EBPSO, MGA, GP, EGP, and MIOST on Example 5

Methods	Feasible Circuits	Average of gates	Average of generations
BPSO	25%	23.95	-
EAPSO	50%	18.65	-
EBPSO	45%	20.1	-
MGA	65%	17.05	-
GP	-	-	-
EGP	60%	9.66	149.5
MIOST	75%	11.6	104.67

Example 6. For our sixth example, (also known as Katz circuit) we have used the truth table shown in Table 12. The parameters used in this example are the following: PS

= 880 and MNG = 4 000 (i.e., a total of 3 520 000 fitness function evaluations). The same values parameters were used by EGP and IWI. As we can see in Table 13, MIOST is the algorithm which has the highest percentage of feasible solutions reached (35%).

Table 12 Truth table of Example 6

a	b	c	d	O_1	O_2	O_3
0	0	0	0	1	0	0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	0	0	1
0	1	0	1	1	0	0
0	1	1	0	0	1	0
0	1	1	1	0	1	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	1	0	0
1	0	1	1	0	1	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	1	0	0

Table 13 Comparison of results among BPSO, EAPSO, EBPSO, MGA, GP, EGP, and MIOST on Example 6

Methods	Feasible circuits	Average of gates	Average of generations
BPSO	-	-	-
EAPSO	-	-	-
EBPSO	-	-	-
MGA	-	-	-
EGP	30%	-	-
MIOST	35%	22.16	277.363

5 Conclusions

To analyze and verify the effectiveness of our approach, called MIOST, we have conducted our experiments in the following way.

1) We have proposed and used a graph-GP representation (IWI) and used only mutation operators to study the effects of allowing p symbols in the individuals′ structures;

2) We allowed the use of the traditional crossover operator on the IWI representation and conducted extensive empirical results;

3) The addition of multiple outputs in the individuals has allowed us to study the approach called MIOST. In other words, MIOST is the result of IWI and the additions of outputs in the individuals′ structures.

We have used six evolvable hardware problems of different complexity (plus the 6-bit multiplixer) to carry out our experiments and analysis with the proposed approach. Our results indicate that MIOST has a better overall performance in terms of consistency in reaching feasible solutions. Our approach, however, was not able to improve previously reported results in terms of number of gates. This is due to: 1) our approach is not an optimization technique, and 2) our approach has the restriction that one or more output depends on the solution of one or more outputs. This can be seen easily by analyzing Fig. 2.

Acknowledgement

The author wish to thank Dr. Katya Rodriguez-Vazquez of IIMAS-UNAM in Mexico City, Mexico who contributed to early versions of this work. Thanks are also due to the reviewers and to professor Edward Tsang for their helpful feedback on an ealier draft of this work. Finally, the author would like to thank the Mexican Consejo Nacional de Ciencia y Tecnologia (CONACyT) for support to pursue postgraduate studies at University of Essex, UK.

References

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

[2] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press, Cambridge, Massachusetts, USA, pp. 154–163, 1994.

[3] P. J. Angeline, J. Pollack. Evolutionary Module Acquisition. In *Proceedings of the 2nd Annual Conference on Evolutionary Programming*, La Jolla, CA, USA, pp. 154–163, 1993.

[4] P. J. Angeline, J. B. Pollack. Coevolving High-level Representations, Technical Report 92-PA-COEVOLVE, Laboratory for Artificial Intelligence, The Ohio State University, USA, 1993.

[5] D. J. Montana. Strongly Typed Genetic Programming, Technical Report 7866, Bolt Beranek and Newman, USA, 1994.

[6] A. Teller, M. Veloso. PADO: Learning Tree Structured Algorithms for Orchestration into an Object Recognition System, Technical Report CS-95-101, Department of Computer Science, Carnegie Mellon University, USA, 1995.

[7] R. Poli. Parallel Distributed Genetic Programming. *New Ideas in Optimization*, D. Corne, M. Dorigo, F. Glover (eds.), McGraw-Hill Ltd., UK, pp. 403–432, 1999.

[8] P. J. Angeline. Multiple Interacting Programs: A Representation for Evolving Complex Behaviours. *Cybernetics and Systems*, vol. 29, no. 8, pp. 779–805, 1998.

[9] J. F. Miller, P. Thomson. Cartesian Genetic Programming. In *Proceedings of European Conference on Genetic Programming, Lecture Notes in Comupter Science*, Springer-Verlag, Edinburgh, Scotland, vol. 1802, pp. 121–132, 2000.

[10] J. F. Miller. An Empirical Study of the Efficiency of Learning Boolean Functions Using a Cartesian Genetic Programming Approach. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, Orlando Florida, USA, vol. 2, pp. 1135–1142, 1999.

[11] W. Kantschick, W. Banzhaf. Linear-tree GP and Its Comparison with Other GP Structures. In *Proceedings of the 4th European Conference on Genetic Programing, Lecture Notes in Computer Science*, Springer-Verlag, Lake Como, Italy, vol. 2038, pp. 302–312, 2001.

[12] W. Kantschick, W. Banzhaf. Linear-graph GP: A New GP Structure. In *Proceedings of the European Conference on Genetic Programming, Lecture Notes in Comupter Science*, Springer-Verlag, Kinsale, Ireland, vol. 2278, pp. 83–92, 2002.

[13] M. Kimura. *The Neutral Theory of Molecular Evolution*, Cambridge University Press, Cambridge, UK, 1983.

[14] E. Galvan-Lopez, R. Poli. An Empirical Investigation of How and Why Neutrality Affects Evolutionary Search. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, ACM Press, Seattle, Washington, USA, pp. 1149–1156, 2006.

[15] E. Galvan-Lopez, R. Poli. Some Steps Towards Understanding How Neutrality Affects Evolutionary Search. In *Proceedings of 9th International Conference on Parallel Problem Solving from Nature, Lecture Notes in Computer Science*, Springer-Verlag, Reykjavik, Iceland, vol. 4193, pp. 778-787, 2006.

[16] R. Poli, E. Galvan-Lopez. On the Effects of Bit-wise Neutrality on Fitness Distance Correlation, Phenotypic Mutation Rates and Problem Hardness. In *Proceedings of Conference on Foundations of Genetic Algorithms, Lecture Notes in Computer Science*, Springer-Verlag, Mexico City, Mexico, vol. 4436, pp. 138-164, 2007.

[17] M. Toussaint. On the Evolution of Phenotypic Exploration Distributions. In *Proceedings of Conference on Foundations of Genetic Algorithms*, Morgan Kaufmann, pp. 169–182, 2003.

[18] C. A. C. Coello, A. H. Aguirre. Design of Combinational Logic Circuits Through and Evolutionary Multiobjective Optimization Approach. *Artificial Intelligence for Engineering, Design, Analysis and Manufacture*, vol. 16, no. 1, pp. 39–53, 2002.

[19] X. Xu, R. C. Eberhart, Y. Shi. Swarm Intelligence for Permutation Optimization: A Case Study on N-Queens Problem. In *Proceedings of IEEE Conference on Swarm Intelligence Symposium*, Hawaii, USA, pp. 243–246, 2003.

[20] E. H. Luna, C. C. A. Coello, A. H. Aguirre. On the Use of a Population-based Particle Swarm Optimizer to Design Combinational Logic Circuits. In *Proceedings of IEEE NASA/DoD Conference on Evolvable Hardware*, Seattle, Washington, USA, pp. 183–190, 2004.

[21] E. Galvan-Lopez, R. Poli, C. A. C. Coello. Reusing Code in Genetic Programming. In *Proceedings of the European Conference on Genetic Programming, Lecture Notes in Comupter Science*, Springer-Verlag, Coimbra, Portugal, vol. 3003, pp. 359–368, 2004.

Edgar Galvan-Lopez received his B. Sc. in computer science from University of Veracruz, Mexico, in 1999. He received his M. Sc. in artificial intelligence from University of Veracruz, Mexico, in 2001. He is currently a Ph. D. candidate in the Deparment of Computing and Electronic Systems at the University of Essex, UK.

His research interests include evolutionary computation systems, mainly in the paradigms of genetic algorithms and genetic programming.