Beneficial Aspects of Neutrality in GP

Edgar Galván López
University of Essex
Colchester, CO4 3SQ, UK
edgar.galvan@gmail.com

Katya Rodríguez Vázquez
IIMAS-UNAM Delegación
Álvaro Obregón México, D.F.
01000, MEXICO
katya@uxdea4.iimas.unam.mx

Riccardo Poli
University of Essex
Colchester, CO4 3SQ, UK
rpoli@essex.ac

ABSTRACT

In this paper we propose a new approach, called Multiple Outputs in a Single Tree (MOST), to Genetic Programming. The idea of this approach is to specify explicitly Neutrality and study how this improves the evolutionary process. For this sake, we have used several evolvable hardware problems of different complexity taken from the literature. Our results indicate that our approach has a better overall performance in terms of consistency to reach feasible solutions.

Keywords: Multiple Outputs in a Single Tree, Neutrality, Evolvable Hardware.

1. INTRODUCTION

Natural Selection is the only theory known which can explain the existence of adaptation in nature. However, this does not mean that natural selection is the only force that directs evolution. Indeed, at molecular scale there is support for the idea that most evolutive variations are neutral. This Neutral Theory has been defined by Motoo Kimura in the late 1970s [6]. The neutral theory does not affirm that during evolution the genes are not making something useful, rather it suggests that different forms of the same gene are indistinguishable in their effects. In other words, a mutation from one gene to another is neutral if this modification does not affect the phenotype.

Kimura's theory is sometimes considered to contradict the Darwinian theory, which is an error. The darwinian theory judges genes by their phenotypic expression whereas Kimura's theory argues that mutations occurring during evolution are neither advantageous nor disadvantageous to the survival and reproduction of individuals. Such random genetic drift should be considered in the study of the evolutionary process.

Some researchers have been interested in understanding how neutrality works in Evolutionary Computation (EC) systems in order to add elements to the evolutionary process to evolve complex problem solutions.

The aims of this paper is to introduce a new-technique that allows one to understand how neutrality works in the evolutionary search process. For the purpose of illustrating this technique we have used several evolvable hardware problems of different complexity taken from the literature. The approach described in this paper has two advantages: (a) it allows specifying explicit neutrality, (b) it makes easier to understand how explicit neutrality improves the evolutionary search process.

The paper is organized as follows. In Section 2, neutrality is explained in the Genetic Programming (GP) paradigm. In Section 3, previous work on neutrality is presented. Section 4 explains the approach proposed. In Section 5, a comparison of results is shown. Analysis and discussion of results are presented in Section 6 and in Section 7 conclusions are drawn.

2. NEUTRALITY

In GP [7, 8], neutrality is often identified with redundancy and introns. Both have being widely studied in the EC community [1, 15, 13, 14, 9, 16].

Functional redundancy refers to the fact that many different individuals, at the phenotype level, represent the same function. For example, the following two genotypes represent the XOR function:

(NOR (AND (NOT (NOT A)) B) (NOT (OR A B)))
(NOR ((NAND (NAND A B) (OR A B)) (NOT (OR A B))))

Introns refer to code that is part of an individual but that semantically does not affect the program's behavior.

The problem with functional redundancy and introns is that both emerge and vary during the evolutionary process and for this reason it is difficult to measure neutrality. The idea of this paper is to specify explicit neutrality and study its effects. For that purpose, we have adopted several combinational logic circuits with multiple outputs. In our approach, as will be explained later, we add an extra element in the function set which will allow explicit neutrality and inactive code.

At this point many questions arise: Is it possible to measure the neutrality in the evolutive process? Does neutrality play an important role during the evolutive process? If so, how much neutrality do we need in our genotype? Which are the effects of neutrality in the evolutive process? We illustrate how we can investigate these research questions with the approach proposed in this paper.

3. PREVIOUS WORK

EC systems are mostly inspired from the theories of genetic inheritance and natural selection. However, neutrality has interested many researchers who want to understand it so that they can incorporate it in their EC systems to solve complex problems.

As we will see in the next paragraphs, the neutrality theory has been explored in Genetic Algorithms (GAs). However, neutrality could be easier to find in GP, due to its representation. In the next paragraphs we will summarize previous work related to neutrality.

Banzhaf proposed an approach where a Genotype-Phenotype Mapping (GPM) [2] was used in the context of constrained optimization problems. In constrained optimization, the solution is not only judged according to its fitness or quality but also has to obey certain restrictions. Banzhaf argued that, very often, constraining the solution space leads to local optima which are difficult to escape with traditional methods. He used high variability of neutral variants to escape from local optima on saddle surfaces. Banzhaf *et al.* extended this previous work in [5].

Maley and Forrest analyzed neutral and selective mutation in cancer [11]. They used an approach based on 2-dimensional cellular automata in which they modeled the evolution of pre-cancerous cells. According to their experiments, one neutral and two selective mutations are necessary and sufficient for the development of cancer. All of the analyses suggest that neutral mutations are the bottleneck in the development of cancer.

Yu and Miller [18] showed in their work how neutrality improves the evolutionary search process for a Boolean benchmark problem. They used Miller's approach called Cartesian GP [12] (CGP) to measure explicit neutrality in the evolutionary process. In their work, Yu and Miller have explained that mutation on a genotype that has part of its genes active and others inactive produces different effects. They argued that mutation on active genes is adaptive because it exploits accumulated beneficial mutations. On the other hand, mutation on inactive genes has neutral effect on a genotype's fitness, yet it provides exploratory power by maintaining genetic diversity.

CGP uses an integer string graph representation. Neutrality between two genotypes with the same fitness is measured by their Hamming distance. Yu and Miller explained in their work that Hamming distance 0 indicates that neither implicit nor explicit neutrality is allowed to be present during evolution. When Hamming distances greater than 0 are allowed, neutrality is allowed to be present, i.e. evolution can proceed in the presence of genetic drift where no fitness improvement occurs. The larger the permitted Hamming distance range is, the larger the amount of neutral mutation allowed during evolution is. The experimental results show that a larger amount of neutrality is better on a Boolean benchmark problem. They argued that with more freedom of genetic drift, evolution generates fitter offspring, i.e. neutrality improves evolvability.

Yu and Miller extended this work in [20]. They formulated the question: Can neutral mutations (those that are neither advantageous nor disadvantageous) benefit evolutionary search?. They answered this question by showing that there is a relationship between neutral mutations and success rate. (They have devised a GP system that utilizes neutrality to search for problem solutions. When applied to a Boolean function problem, the solutions were found with a higher success rate.)

In their experiments reported in [20], the amount of neutral mutations is measured in the selection step, which evaluates both the fitness and the number of neutral mutations in the evolved entities. They conclude the following based on their results: (a) neutrality does not impact on success rates when the number of solutions in the search space is large, (b) however, when the difficulty level is slightly increased any amount of neutrality improves the success rates for rates of 1% or 2%, (c) increasing mutation rates also improves success rates, and (d) for more difficult problems (even-12-parity), high neutrality and mutation rates are not sufficient to find a solution.

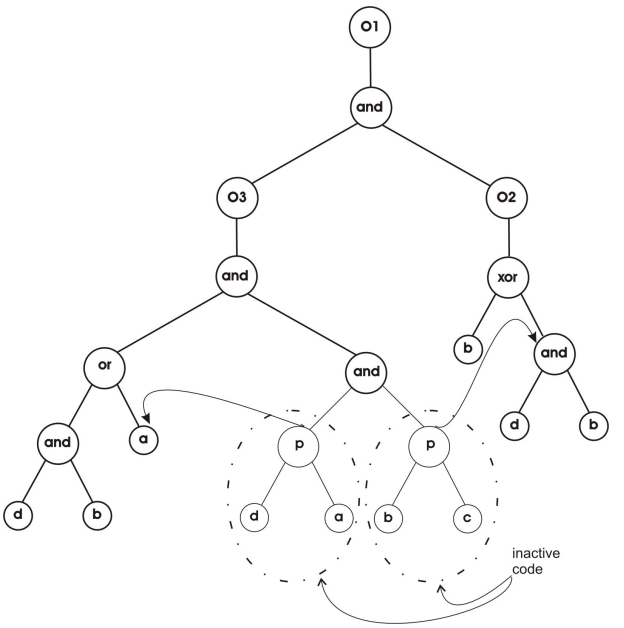

Figure 1: Example of an individual using *p* symbols as functions. We have Pure Neutrality when a *p* symbol is pointing to any node which is not part of a *p* symbol's subtree.

Yu and Miller also investigated neutrality using the simple OneMax problem [19, 21]. They attempted a theorical approach in this work. With their experiments, they showed that neutrality is advantageous because it provides a buffer to absorb destructive mutations.

4. APPROACH

Our approach, which we denominate *Multiple Outputs in a Single Tree* (MOST), will allows us to study the effects of neutrality in Boolean benchmark problems. This new approach was inspired by the EGP approach [4] which allows reusing code and also evolving graph-like structures, which could encode, for instance, combinational logic circuits.

In MOST, a program is represented as a tree as suggested by Koza [7] with the main difference that a program has as many outputs as the circuit to solve has. For example, Figure 1 shows a typical MOST individual, which has 3 outputs represented by O1, O2 and O3. In this approach the function set included Boolean functions {*AND, OR, XOR, NOT*}, while the terminal set included letters {*a, b, c, ...*} representing input wires. We also consider another set which contain as many ouputs as the circuit to solve has. So, when we create an individual, we choose randomly from the set of outputs any of these and we eliminate it from this set. Once, we have created an individual, we check if it contains all the outputs, if not we repeat the process until we have created a valid individual. It is worth mentioning that when we create an individual we do not specify which output will be in the root, so GP has the power to place the most complex output in the root (results confirm this statment). Once an individual has been created, we use a probability to replace

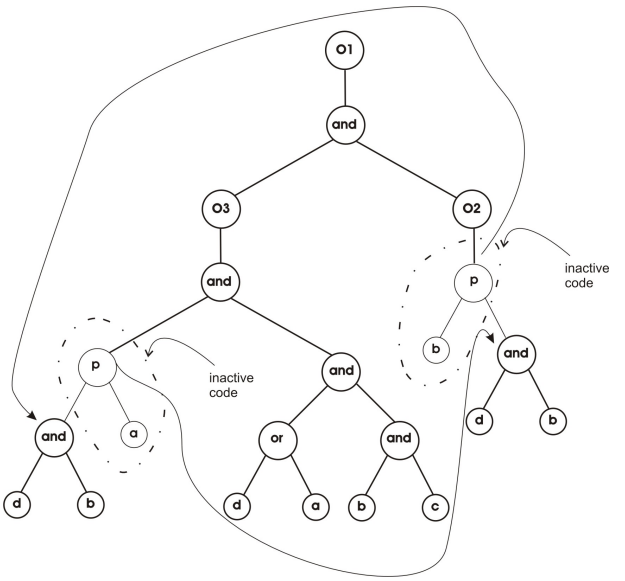

Figure 2: Example of an individual using p function symbols. We have Impure Neutrality when a p symbol points to a node which is contained inside a tree rooted at p symbol.

a function with a p symbol which is a function of arity 2. This p symbol will allow us specify explicit neutrality. The p symbol will point to code somewhere else in the program. So, when a p is executed, the subtree rooted at that node is ignored, and so plays the role of inactive code.

In Figure 1 we show an individual with 22 nodes, 16 of which are active and 6 inactive. This is a case of *Pure Neutrality*, that is defined as a p symbol pointing to any node which is not part of the tree rooted at the p symbol itself or another p symbol. As we have explained earlier, the p symbol can point to any part of the individual, so the p symbol could point to a node which is contained inside the tree rooted at the p symbol. If this is the case, we denominate this *Impure Neutrality*. In Figure 2 we can see an example of impure neutrality. In this case, there are 18 active nodes and 4 inactive nodes. Of course, a combination of pure and impure neutrality can be present in the same individual.

The crossover operator used in MOST works as usual but an important difference is that, if the sub-tree swapped contained a p symbol, the p symbol's pointer is not changed.[1] Another difference is that once we have created our initial population, we classify each node of each individual in order to know which nodes can be used to apply crossover. With this we assure that an individual will contain the number of outputs that must contain to solve the circuit. This classification is only applied when the circuit to solve has more than one output. 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.

The fitness function that we used for circuit design works in two stages: at the beginning of the search, the fitness of a genotype is the number of correct output bits (raw fitness). Once the fitness has reached the maximum number of correct outputs bits, we try to optimize the circuits by giving a higher fitness to individual with shorter encodings.

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

Table 1: Truth table of the first example.

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

Table 2: Truth table of the second example.

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

5. COMPARISON OF RESULTS

We used several evolvable hardware problem of different complexity taken from the literature to test MOST. Our results were compared with those obtained by MGA [3], by EAPSO [17], by EBPSO [10], by BPSO [10], by EGP and by traditional GP. For all the examples, we performed 20 independent runs using the following set of gates: $\{AND, OR, XOR, NOT\}$. After a series of preliminary experiments we have decided to use a crossover rate of 70%, a mutation rate of 0.02% and p rate of 0.01 for all the experiments, except for example 4 and 5, where the p rate is 0.08.

As we will see in the next subsections, in all the experiments we improve the percentage of feasible region[2] compared against the other techniques.

5.1 Example 1

For our first example we have used the truth table shown in Table 1. The parameters used in this example are the

[2]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: Truth table of the third example.

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 4: Truth table of the fourth example.

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 5: Truth table of the fifth example.

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 6: Comparison of results between BPSO, EAPSO, EBPSO, MGA, GP, EGP and MOST on the first example.

	Feasible circuits	*Avg. # of gates*	*Avg. of gen.*
BPSO	100%	6.75	-
EAPSO	95%	7.3	-
EBPSO	100%	6.15	-
MGA	90%	9.3	-
GP	90%	11.05	56.05
EGP	-	-	-
MOST	100%	10.4	35.61

following: Population Size (PS) = 190 and the Maximum Number of Generations (MNG) = 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 6, the only algorithms able to converge to a feasible region in 100% of the runs performed were BPSO, EBPSO and MOST. Moreover, in MOST 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 MOST was 10.4, while the average number of gates in EBPSO was 6.15.

5.2 Example 2

For our second example we have used the truth table shown in Table 2. The parameters used in this example are the following: PS = 240 and the MNG = 415 (i.e., a to-

Table 7: Comparison of results between BPSO, EAPSO, EBPSO, MGA, GP, EGP and MOST on the second example.

	Feasible circuits	*Avg. # of gates*	*Avg. of gen.*
BPSO	85%	10.4	-
EAPSO	90%	8.25	-
EBPSO	100%	5.9	-
MGA	70%	13.7	-
GP	90%	9.22	49.23
EGP	-	-	-
MOST	100%	8.6	28.88

Table 8: Comparison of results between BPSO, EAPSO, EBPSO, MGA, GP, EGP and MOST on the third example.

	Feasible circuits	*Avg. # of gates*	*Avg. of gen.*
BPSO	-	-	-
EAPSO	50%	13.8	-
EBPSO	55%	12.15	-
MGA	25%	21.4	-
GP	5%	90	791
EGP	-	-	-
MOST	70%	32.28	156.57

Table 9: Comparison of results between BPSO, EAPSO, EBPSO, MGA, GP, EGP and MOST on the fourth example.

	Feasible circuits	*Avg. # of gates*	*Avg. of gen.*
BPSO	95%	10.05	-
EAPSO	70%	13.45	-
EBPSO	100%	7.75	-
MGA	75%	13.4	-
GP	-	-	-
EGP	55%	9.7	122.9
MOST	100%	12.9	109.55

Table 10: Comparison of results between BPSO, EAPSO, EBPSO, MGA, GP, EGP and MOST on the fifth example.

	Feasible circuits	*Avg. # of gates*	*Avg. of gen.*
BPSO	25%	23.95	-
EAPSO	50%	18.65	-
EBPSO	45%	20.1	-
MGA	65%	17.05	-
GP	-	-	-
EGP	60%	9.66	149.5
MOST	75%	11.6	104.67

tal 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 7, the only algorithms able to converge to a feasible region in 100% of the runs performed were EBPSO and MOST. Moreover, in MOST the average of generations required to solve the circuit was 28.88, while in GP the average of generations was 49.23. However, the average number of gates in MOST was 8.6, while the average number of gates in EBPSO was slightly smaller (5.9).

5.3 Example 3

For our third example we have used the truth table shown in Table 3. The parameters used in this example are the following: PS = 550 and the 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 8, MOST is the algorithm with the highest percentage of feasible solutions reached (70%). Moreover, in MOST the average of generation at which it solved the circuit was 156.57, while in GP it was 791. However, the average number of gates in MOST was 32.28, while the average number of gates in EBPSO was much smaller (12.15).

5.4 Example 4

For our fourth example we have used the truth table shown in Table 4. The parameters used in this example are the following: PS = 380 and the MNG = 525 (i.e., a total of 199,500 fitness function evaluations). The same values parameters were used by EGP. 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 MOST. Moreover, in MOST the average of generations at which it solved the circuit was 109.55, while in EGP it was 122.9. However, the average number of gates in MOST was 12.1, while the average number of gates in EBPSO was 7.75 gates.

5.5 Example 5

For our fifth and last example we have used the truth table shown in Table 5. The parameters used in this example are the following: PS = 1,200 and the 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 10, MOST is the algorithm which has the highest percentage of feasible solutions reached (75%). Moreover, in MOST the average of generations at which it solved the circuit was 104.67, while in EGP it was 149.5. Surprising, MOST is one of the algorithms with lowest average number of gates with 11.6, only behind EGP with 9.66 gates.

6. ANALYSIS AND DISCUSSION OF RESULTS

In the previous section we have shown results of several circuits of different degrees of complexity. We will analyze only the last 3, which are the most difficult, due to space

limitations. However, it is worth mentioning that all results show a similar behavior.

On the left of Figure 3, we can see the average of active and inactive nodes (over all runs) and on the right we can see the average fitness. As we can observe on the left of the figure, at the beginning the number of inactive nodes tends to increase and after in few generations it tends to decrease. The same behavior can be seen with active nodes. If we analize the fitness plot, we can see how the search does not get stuck in a local optimum. In other words, fitness tends to increase constantly. For instance, the first 193 generations show this tendency it can be seen that inactive nodes are present during these generations. This fact gives us the idea that neutrality favors the evolutionary process to reach the feasible region. In other words, neutrality seems to smooth the landscape.

On the left of Figure 4, we can see the average of active and inactive nodes (overall runs) and on the right, we can see the average fitness. As we can observe, on the left of the figure, at the beginning the number of inactive nodes tends to increase and after in few generations tends to decrease. The same behavior can be seen with active nodes. If we analize the right figure, we can see how the search does not get stuck in a local optimum. For instance, the first 113 generations show this tendency it can be seen that inactive nodes are present during these generations.

On the left of Figure 5, we can see the average of active and inactive nodes (over all runs)in the right, we can see the average fitness. As we can observe, on the left of the figure, at the beginning the number of inactive nodes tends to increase and after in few generations it tends to decrease. The same behavior can be seen with active nodes. If we analize the fitness plot, we can see how the search does not get stuck in a local optimum. For instance, the first 79 generations show this tendency it can be seen that inactive nodes are present during these generations.

7. CONCLUSIONS

Neutrality is an important research subject. Neutrality is identified with redundancy and introns. However, it is difficult to measure these since they emerge and vary during the evolutionary process. With the use of the approach described in this paper, we have allowed explicit neutrality in the evolutionary process. This has allowed us to study the effects of neutrality during the evolutionary process.

Our results indicate that our proposed approach has a better overall performance in terms of consistency in reaching feasible solutions. However, our approach was not able to improve previously published results in terms of number of gates. This is due two main factors: (a) our approach is not an optimization technique and, (b) our approach has the restriction that one or more outputs depend on the solution of one or more outputs. This can be seen easily analyzing figure 1.

The results reported in Section 6 are an illustration of the power of this technique. From these we can conclude that for the benchmark Boolean problems we studied:

- Neutrality has beneficial effects when it is present with low probability, as for example, p= 0.01 for easier problems and p = 0.08 for harder problems.
- Neutrality does not allow to get stuck in a local optimum.
- Neutrality helps the evolutionary process reach the feasible region. In other words, it smoothes the landscape.

Acknowledgments

The first author acknowledges computing facilities at IIMAS-UNAM. The second author gratefully acknowledges support from CONACyT through project 40602-A.

8. REFERENCES

[1] P. J. Angeline. Genetic programming and emergent intelligence. In K. E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 4, pages 75–98. MIT Press, 1994.

[2] W. Banzhaf. Genotype-phenotype-mapping and neutral variation – A case study in genetic programming. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature III*, volume 866 of *LNCS*, pages 322–332, Jerusalem, 9-14 Oct. 1994. Springer-Verlag.

[3] C. A. C. Coello and A. H. Aguirre. Design of combinational logic circuits through and evolutionary multiobjective optimization approach. In *Artificial Intelligence for Engineering, Design, Analysis and Manufacture*, volume 16, pages 39–53, January. 2002.

[4] E. Galvan Lopez, R. Poli, and C. C. Coello. Reusing code in genetic programming. In M. Keijzer, U.-M. O'Reilly, S. M. Lucas, E. Costa, and T. Soule, editors, *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 359–368, Coimbra, Portugal, 5-7 Apr. 2004. Springer-Verlag.

[5] R. E. Keller and W. Banzhaf. Genetic programming using genotype-phenotype mapping from linear genomes into linear phenotypes. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 116–122, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[6] M. Kimura. Evolutionary rate at the molecular level. In *Nature*, volume 217, pages 624–626, 1968.

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

[8] W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer, 2002.

[9] S. Luke. Code growth is not caused by introns. In D. Whitley, editor, *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, pages 228–235, Las Vegas, Nevada, USA, 8 July 2000.

[10] E. H. Luna, C. C. Coello, and A. H. Aguirre. On the use of a population-based particle swarm optimizer to design combinational logic circuits. In R. S. Zebulum, D. Gwaltney, G. Hornby, D. Keymeulen, J. Lohn, and A. Stoica, editors, *Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware*, pages 183–190, Los Alamitos, California, June 2004. IEEE Computer Society.

[11] C. C. Maley and S. Forrest. Modeling the role of neutral and selective mutations in cancer. In *Artificial*

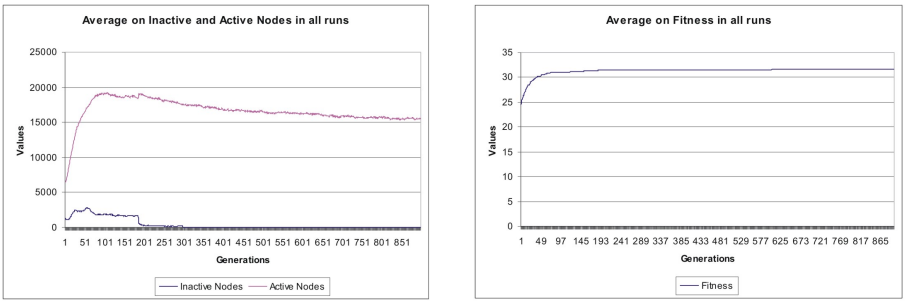

Figure 3: Average of the number of active and inactive nodes per generation of all runs, with 0.01 p rate (left) and average of fitness of all runs (right) on example 3.

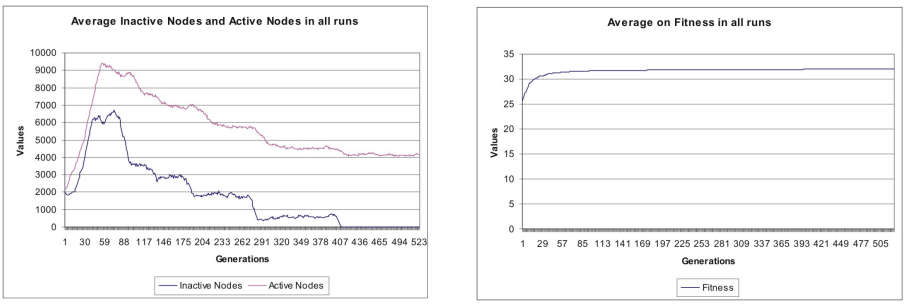

Figure 4: Average of the number of active and inactive nodes per generation of all runs, with 0.08 p rate (left) and average of fitness of all runs (right) on example 4.

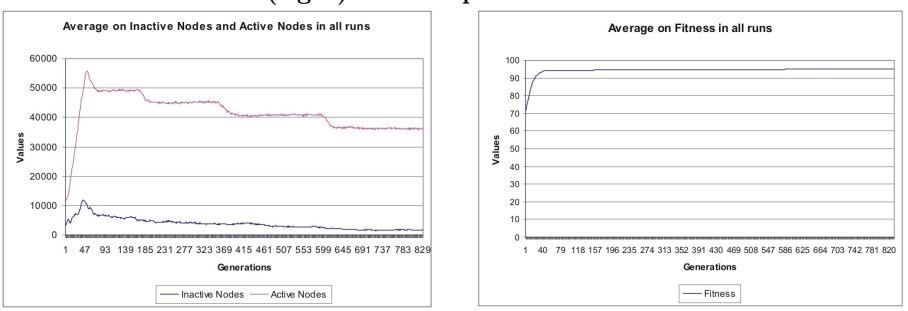

Figure 5: Average of the number of active and inactive nodes per generation of all runs, with 0.08 p rate (left) and average of fitness of all runs (right) on example 5.

Life VII: Proceedings of the Seventh International Conference, pages 395–404. MIT press, 2000.

[12] J. F. Miller and P. Thomson. Cartesian genetic programming. In R. Poli, W. Banzhaf, W. B. Langdon, J. F. Miller, P. Nordin, and T. C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 121–132, Edinburgh, 15-16 Apr. 2000. Springer-Verlag.

[13] P. Nordin and W. Banzhaf. Complexity compression and evolution. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 310–317, Pittsburgh, PA, USA, 15-19 July 1995. Morgan Kaufmann.

[14] P. Nordin, W. Banzhaf, and F. D. Francone. Introns in nature and in simulated structure evolution. In D. Lundh, B. Olsson, and A. Narayanan, editors, *Bio-Computation and Emergent Computation*, Skovde, Sweden, 1-2 Sept. 1997. World Scientific Publishing.

[15] P. Nordin, F. Francone, and W. Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In J. P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 6–22, Tahoe City, California, USA, 9 July 1995.

[16] T. Soule. Exons and code growth in genetic programming. In J. A. Foster, E. Lutton, J. Miller, C. Ryan, and A. G. B. Tettamanzi, editors, *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, volume 2278 of *LNCS*, pages 142–151, Kinsale, Ireland, 3-5 Apr. 2002. Springer-Verlag.

[17] X. Xu, R. C. Eberhart, and Y. Shi. Swarm intelligence for permutation optimization: A case study on n-queens problem. In *Proceedings of the IEEE Swarm Intelligence Symposium 2003 (SIS 2003)*, pages 243–246, Indianapolis, Indiana, USA, . 2003.

[18] T. Yu and J. Miller. Neutrality and the evolvability of boolean function landscape. In *Fourth European*

Conference on Genetic Programming, pages 204–211. Springer-Verlag, 2001.

[19] T. Yu and J. Miller. Climbing unimodal landscapes with neutrality: A case study of the one-max problem. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, page 704, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[20] T. Yu and J. F. Miller. Needles in haystacks are not hard to find with neutrality. In J. A. Foster, E. Lutton, J. Miller, C. Ryan, and A. G. B. Tettamanzi, editors, *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, volume 2278 of *LNCS*, pages 13–25, Kinsale, Ireland, 3-5 Apr. 2002. Springer-Verlag.

[21] T. Yu and J. F. Miller. The role of neutral and adaptive mutation in an evolutionary search on the onemax problem. In E. Cantú-Paz, editor, *Late Breaking Papers at the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pages 512–519, New York, NY, July 2002. AAAI.