Towards Understanding the Effects of Locality in GP

Edgar Galván-López,
Michael O'Neill and
Anthony Brabazon
Natural Computing Research & Applications Group,
Complex and Adaptive Systems Lab
University College Dublin
Email: {edgar.galvan, m.oneill, anthony.brabazon}@ucd.ie

***Abstract*—Locality - how well neighbouring genotypes correspond to neighbouring phenotypes - has been defined as a key element in Evolutionary Computation systems to explore and exploit the search space. Locality has been studied empirically using the typical Genetic Algorithms (GAs) representation (i.e., bitstrings), and it has been argued that locality plays an important role in the performance of evolution. To our knowledge, there are no studies of locality using the typical Genetic Programming (GP) representation (i.e., tree-like structures). The aim of this paper is to shed some light on this matter by using GP. To do so, we use three different types of mutation taken from the specialised literature. We then perform extensive experiments by comparing the difference of distances at the genotype level between parent and offspring and their corresponding fitnesses. Our findings indicate that there is low-locality in GP when using these forms of mutation on a multimodal-deceptive landscape.**

I. Introduction

The concept of a fitness landscape was first introduced in biology by [18]. This concept has dominated the way geneticists think about biological evolution and has been adopted within the Evolutionary Computation (EC) community. In simple terms, a fitness landscape can be seen as a plot where each point on the horizontal axis represents all the genes in an individual corresponding to that point. The fitness of that individual is plotted as the height against the vertical axis. Thus, a fitness landscape is a representation of a search space which may contain peaks, valleys, hills and plateaus.

How an algorithm explores and exploits such a landscape is a key element during evolutionary search. In [14], [15], Rothlauf mentioned that locality (i.e., how well neighboring genotypes correspond to neighboring phenotypes) must be taken into account in EC systems. The author distinguished two forms of locality: low and high locality. The author pointed out that a representation presents high locality if all neighboring genotypes correspond to neighboring phenotypes. On the other hand, a representation presents low locality if some neighboring genotypes do not correspond to neighboring phenotypes. The author also mentioned that a representation that has high locality is necessary for efficient evolutionary search (in Section II we further explain the concept of locality).

In his original studies, Rothlauf used Genetic Algorithms to conduct his experiments [13]. To our knowledge, we have not seen any studies on locality using the typical Genetic Programming (GP) [7], [11] representation (i.e., tree-like structures). The goal of this paper is to shed some light on the type of locality present in GP.

This paper is organised as follows. In the next section, locality in EC is summarised. In Section III, we describe a well-studied distance metric for tree-like structures. In Section IV, we describe how we study locality at the genotype-fitness level in GP. In Section V, we present and discuss our findings. Finally, in Section VI we draw some conclusions.

II. Low and High Locality

In [14], [15], Rothlauf stated that the understanding of how well neighbouring genotypes correspond to neighbouring phenotypes is a key element in evolutionary search.

To study locality, it is necessary to define a metric on the search space Φ. In a genotype-phenotype mapping representation, it is clear that Φ_g is the genotypic search space and Φ_p is the phenotypic search space. Now, based on a defined metric we can quantify how different or similar two individuals are. In his work, Rothlauf claimed that for two different search spaces (e.g., genotypic and phenotypic search space) it is necessary to define two different metrics. In Sections III and IV we further discuss this.

The author distinguished two types of locality: low and high locality. The author pointed out that a representation presents high locality if all neighboring genotypes correspond to neighboring phenotypes. On the other hand, a representation presents low locality if some neighboring genotypes do not correspond to neighboring phenotypes. The author also mentioned that a representation that has high locality is necessary for efficient evolutionary search.

Rothlauf stated that if a representation shows high locality then any search operator has the same effects in both the genotype and phenotype space. It is clear then that the difficulty of the problem remains unchanged.

This, however, changes when a representation has low locality. To explain how low locality affects evolution, Rothlauf considered three different categories[1]. These are:

- *easy*, in which fitness increases as the global optimum approaches,
- *difficult*, for which there is no correlation between fitness and distance and

[1]These categories were taken from the work presented in [6].

- *misleading*, in which fitness tends to increase with the distance from the global optimum.

If a given problem lies in the first category (i.e., easy), a low locality representation will change this situation by making it more difficult and now, the problem will lie in the second category. According to the author, this is due to low locality randomising the search. To explain this, Rothlauf mentioned that representations with low locality lead to uncorrelated fitness landscapes, so it is difficult for heuristics to extract information.

If a problem lies in the second category, this type of representation does not change the difficulty of the problem. The author pointed out that there are representations that can convert a problem from difficult (class two) to easy (class one). However, according to the author, few representations have this effect.

Finally, if the problem lies in the third category, a representation with low locality will transform it and now, the problem will lie in the second category. That is, the problem is less difficult because the search has become more random. As can be seen, this is a mirror image of a problem lying in the first category and using a representation that has low locality.

In his work, Rothlauf mentioned *"..., it remains unclear as to how the locality of a representation influences problem difficulty, and if high-locality representations always aid evolutionary search."*[13, page 73]. In this work, we are interested in seeing how locality affects problem difficulty. To do so, it is necessary to define a metric. This will be presented in the following section.

III. Distance in Tree-Like Structures

In this study we examine the traditional tree-based GP representation. As such, we do not explicitly distinguish between genotypic (Φ_g) and phenotypic (Φ_p) search spaces. Instead we examine the relationship between the search space (Φ) and fitness.

When using the traditional GA representation (binary strings), it is straightforward to calculate the distance between two given strings. This is accomplished using the Hamming distance. The Hamming distance between two strings of the same length is the number of positions for which the corresponding symbols are different. This scenario, however, changes radically when using the traditional GP representation.

[1], [16], [17] have shown that calculating distances between tree-like structures is very difficult. Nevertheless, the authors were able to define a distance that has proved to be reliable when it was used to calculate the hardness of a problem [3], [4], [16], [17]. In the following paragraphs, we will see how this distance works.

In [1], the authors proposed a distance to calculate *fdc*. However, as the authors pointed out in their research, to calculate distances at genotype level when using the syntax tree representation is a difficult task. In their investigation, they used the same function and terminal sets defined in [12] where the function set is $F_{set} = \{A, B, C, \cdots\}$, where A is a function of arity 1, B has arity 2, C has arity 3 and so forth. The terminal set was composed of a single symbol X. Given the difficulties in calculating distances between tree-like structures, Clergue and co-workers started their studies with a restriction: the nodes below the current node with arity n cannot have an arity equal to or greater than n. They called this representation *limited* GP.

Initially, Clergue et al. defined the distance between two trees as follows:

Let T_1 and T_2 be two trees. Then,

$$d_1(T_1, T_2) = |weight(T_1) - weight(T_2)| \tag{1}$$

where $weight(T) = 1 \cdot n_X(T) + 2 \cdot n_A(T) + 3 \cdot n_B(T) + 4 \cdot n_C(T) + \cdots$ and $n_X(T)$ is the number of X symbols in the tree T, $n_A(T)$ is the number of symbols A in the tree T, $n_B(T)$ is the number of symbols B in the tree T, and so on.

This first attempt to calculate the distance between pairs of trees had a major problem: two trees with very different structures between them can have distance 0.

Making an effort to overcome this problem, they defined a new concept of distance. That is, each tree with root i must have a greater weight than the trees with root j, if and only if $j < i$, where:

1) $i, j \in \{X, A, B, C, \cdots\}$
2) there is an order such that $X < A < B < C \cdots$

Formally, the distance between two trees is given as follows:

$$d_2(T_1, T_2) = |weight(T_1) - weight(T_2)| \tag{2}$$

where $weight(T) = 1 \cdot n_X(T) + 2 \cdot n_A(T) + 3 \cdot n_B(T) + 4 \cdot n_C(T) + \cdots + prize(T)$ and $prize(T)$ is computed as 2 plus the difference between the heaviest tree of root $succ(i)$ (i.e., $succ(C) = B$) and the lightest tree of root (i). For instance, to calculate the prize of root C, we need to compute the heaviest tree of root B (i.e., represented by *(B (A X) (A X))* in prefix notation) whose weight is 9 and the lightest tree of root C (i.e., represented by *(C X X X)* in prefix notation) whose weight is 7. Adding 2 to the difference $9-2$, we obtain the prize for root C, which is 4. Similarly, we have that the prizes for roots D, E and F are 28, 148 and 788, respectively.

Despite "rewarding" individuals, Clergue and co-authors still had a problem: two individuals that are symmetrical about a vertical axis have a distance 0 but their structures are different. As can be seen, calculating distance between trees is a very difficult task indeed.

Finally, they extended their previous distance measures obtaining a new distance which works for all trees. There are three steps to calculate the distance between tree T_1 and T_2:

1) T_1 and T_2 must be aligned to the left-most subtrees (Figure 1 illustrates this idea),
2) For each pair of nodes at matching positions, the difference of their codes c (typically c is the index of an instruction within the primitive set) is calculated, and
3) The differences calculated in the previous step are combined into a weighted sum (nodes that are closer to the root have greater weights than nodes that are at lower levels).

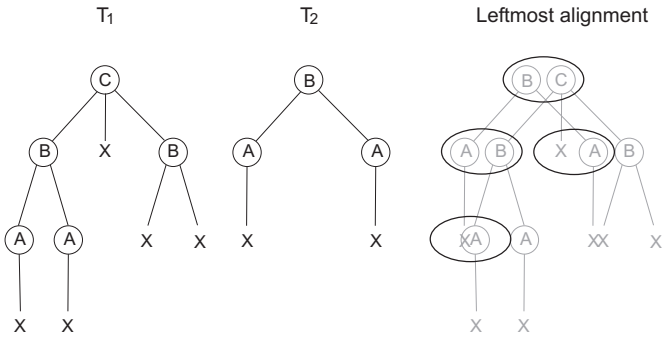

Fig. 1. Alignment for measure *dist*. T_1 and T_2 represent two trees taken from the population (left), the figure at the right shows with thick circles the matching positions when leftmost alignment is applied to trees T_1 and T_2.

Formally, the distance between trees T_1 and T_2 with roots R_1 and R_2, respectively, is defined as follows:

$$dist(T_1,T_2,k) = d(R_1,R_2) + k\sum_{i=1}^{m} dist(child_i(R_1), child_i(R_2), \frac{k}{2}) \quad (3)$$

where: $d(R_1,R_2) = (|c(R_1) - c(R_2)|)^z$ and; $child_i(Y)$ is the i^{th} of the m possible children of a node Y, if $i < m$, or the empty tree otherwise. Note that c evaluated on the root of an empty tree is 0 by convention. The parameter k is used to give different weights to nodes belonging to different levels in the tree and $z \in \mathbb{N}$ is a parameter of the distance.

The distance shown in Equation 3 has successfully proved to be a reliable distance [3], [4], [16], [17]. This is the distance that we will use in our experiments. For this purpose, we will use the code provided in [3] to make such calculations.

IV. Approach

In the specialised literature [11], one can find many different alternatives from the typical GP representation (i.e., tree-like structures) to represent an individual, and there are many different reasons why researchers have proposed alternative representations. For instance, allowing the reuse of code [5], allowing inactive code [9], etc. Nevertheless, GP tree-like structures are the most widely adopted. This form of representation does not consider a genotype-phenotype mapping that is very common in Genetic Algorithms. Of course, there are variations in GP that allows us to study the relationship between the genotype and the phenotype (e.g., [2], [10] and [9]).

Because we are interested in seeing how locality affects GP search, we need to consider how a change in the individual is translated at the fitness level. At this point some questions arise: what kind of relation shall we expect from an individual being affected by an operator and its corresponding fitness? To answer this question we will focus our attention on how different types of mutation affect individuals (both in terms of structure and their corresponding fitnesses).

For our experiments, we have used the Artificial Ant Problem [7, pages 147–155]. This problem consists of finding a program that can successfully navigate an artificial ant along a path of food on a 32 x 32 toroidal grid. This problem has been shown to be difficult for GP for its characteristics (e.g., multimodal-deceptive features)[8, Chapter 9]. These features are believed to be common in many real-world applications.

The terminal set for this problem is $T = \{Move, Right, Left\}$. These operations move the ant forward one square, turn 90° to the right and turn 90° to the left, respectively. When the ant performs any of these operations, it consumes one time unit. The function set is $F = \{IfFoodAhead, Prog2, Prog3\}$. *IfFoodAhead* is a conditional function that executes the first argument if the ant perceives food in front of it, otherwise the second argument is executed. The remaining functions, *Prog*2 and *Prog*3, have arity two and three, respectively. Both functions execute their arguments in sequence.

The problem is in itself challenging for many reasons. The ant must eat all the food pellets (89 in total) along a track that has single, double and triple gaps along it. Moreover, the food trail is twisted. For this task, the ant is given a certain number of steps (normally 600 steps) to find all the food available in the grid. Once the ant finds and eats a food pellet, its fitness increases by one (raw fitness) so, the maximum fitness for this problem is 89.

For our studies we have considered the use of three different mutation operators:

1) subtree mutation replaces a randomly selected subtree with another randomly created subtree [7],
2) one-point mutation replaces a point in the individual subject to a probability (i.e., more than one can be changed or none) and
3) inflate mutation consists of inserting a terminal node beneath a function whose arity a is lower than the maximum arity defined in the function set, then the function is replaced by another function of arity $a+1$ [17].

We know that for the Artificial Ant Problem there are 90 different fitness values (including 0 meaning that an individual did not eat any food). For each of those 90 different values, we generated 100 different individuals. We achieved this by performing thousands of independent runs until finding 100 different individuals for each of the 90 different fitness values (Table I shows the parameters used to capture these individuals). Once this has been done, for each group of 100 individuals (i.e., 9000 individuals in total), we performed 2000 mutations for each operator (i.e., one-point, inflate and subtree mutation) independently, as described previously. In the following section, we will present and discuss our findings.

V. Results and Discussion

In Figures 2, 3 and 4, we show the relation between a new fitness (i.e., fitter individuals or less fit individuals) and compared them with their corresponding distances. Let us start by analysing the proportion of "good" resulting offspring (i.e., fitter individuals) and "bad" resulting offspring (i.e., less fit individuals) after applying a mutation operator (top of Figures 2, 3 and 4).

TABLE I
PARAMETERS FOR THE ANT PROBLEM.

Objective	Find a program that follows the "Santa Fe Ant Trail"
Function set	IfFoodAhead, Prog2, Prog3
Terminal set	Move, Right, Left
Fitness cases	The Santa Fe Ant Trail
Fitness	Food eaten
Selection	Tournament (size 7)
Wrapper	Program repeatedly executed for 600 time steps
Initial Population	Ramped half and half (depth 1 to 6)
Population size	500
Independent runs	100
Operators	One-Point Mutation, Inflate Mutation and Subtree Mutation
Parameters	One mutation per individual
Termination	Maximum number of generations, $G = 50$

We know that mutations are well-known for their destructive effects and, in fact, this is corroborated at the top of these figures. Notice how for any type of the mutation operators used for this problem, the number of fitness improvements is less than the number of disimprovement.

Let us go deeper in our analysis by analysing when an individual is affected by mutation and the resulting offspring is less fit (centre of Figures 2, 3 and 4). The first thing to notice is that by using any of the three mutations explained previously, the resulting fitness can drop dramatically (e.g, from fitness 80-89 to 0-40). The corresponding difference of distance (i.e., from an individual to their produced offspring) is also quite big. So one could suggest that there is a strong relation between distance and fitness. However, when the fitness is in the range of 0-5, the new fitness (now in the range of 1-4) does not correspond with the difference of distances. In fact, for all three mutations one can observe how the highest peak is in this region. This indicates that small variations in the fitness do not necessarily correspond to a small variation of distance.

Now, let us focus our attention when the new fitness is better than the original fitness (bottom of Figures 2, 3 and 4). In the case of inflate mutation (see Figure 2), we can observe that the improvement in fitness does not present big variations (i.e., error bars). This, however, is not the case for one-point mutation (see Figure 3), where there are some cases where a fitness can have quite a big jump. Now, for the subtree mutation (Figure 4), we have that the behaviour presented in the fitness improvement is more less similar compared against inflate mutation. However, there is a big difference when these two mutations are compared. In the case of subtree mutation, one can see that the difference of distance between the parent and the offspring is not big, in contrast with what you can observe with inflate mutation (Figure 2) and one-point mutation (Figure 3). This corresponds to small changes in the improvement of fitness.

Within the frame of locality defined and explained by [13] and using the distance proposed and defined in [1], [16], [17], it is quite clear that the mutations presented here present low locality at the Φ-fitness level. However, low-locality affects evolutionary search in different ways, as explained previously. The most representative case is when one compares subtree mutation and one-point mutation, where it can be seen that small improvement in fitness correspond to small difference of distances for the subtree mutation, whereas the opposite is true for the one-point mutation.

So far, we have shown how locality at the Φ-fitness level is different and in what ways for three forms of mutation presented in Section IV. However, it is not clear what changes each of these mutations induce. In a mutation-based GP system, one can record the changes produced by mutations. In Figure 5, we show the original fitness in the x axis and their corresponding new fitness (y axis). For clarity purposes, we have divided the plots into two categories for each of the mutation operators: less fit individuals (left of Figure 5) and fitter individuals (right of Figure 5). When a new individual is generated by means of a mutation and its corresponding fitness is lower than its parent's fitness, we can see how for all the mutations used, there are many occurrences (note log z axis scale). These occurrences, however, vary. For instance, for the case of subtree mutation we can see a more uniform behaviour, whereas it is the opposite for one-point mutation and inflate mutation. For the latter, this is quite obvious if one observes that there is a single peak between fitnesses 80-89 and drops dramatically (i.e., between 10 and 20) (see top-left of Figure 5).

Now, let us focus our attention when a mutation produces beneficial effects (e.g., improvement of fitness). This is shown in the right-hand side of Figure 5. It can be seen how the same effects shown for a detrimental mutation can be also seen for a beneficial mutation. That is, individuals with very "bad" fitness (e.g., fitness in the region of 0 to 10) can jump up to the region of 80-89 (and it is here where the solution is). This scenario is present for all the mutations used. For subtree mutation, however, this is even more frequent as can be seen in the bottom-right of Figure 5. This is perhaps one of the reasons why by using subtree mutation, a GP system is able to find the solution, as shown in [8].

VI. CONCLUSIONS

We presented an analysis of the locality of three mutation operators for the classic tree-based Genetic Programming representation on the Santa Fe ant trail benchmark problem, which has been shown to have multimodal-deceptive features [8, Chapter 9]. These features are believed to be common on many real-world applications and that is why the problem analysed in our work is significant.

Results indicate that the mutation operators examined are inconsistent with respect to the quality of locality as measured by fitness and structural changes. By carrying out a deeper analysis on the three mutation operators used to conduct our experiments, we have shown how each of the three operators exhibits relatively low-locality, however each operator samples new points differently in the search space. Further research is

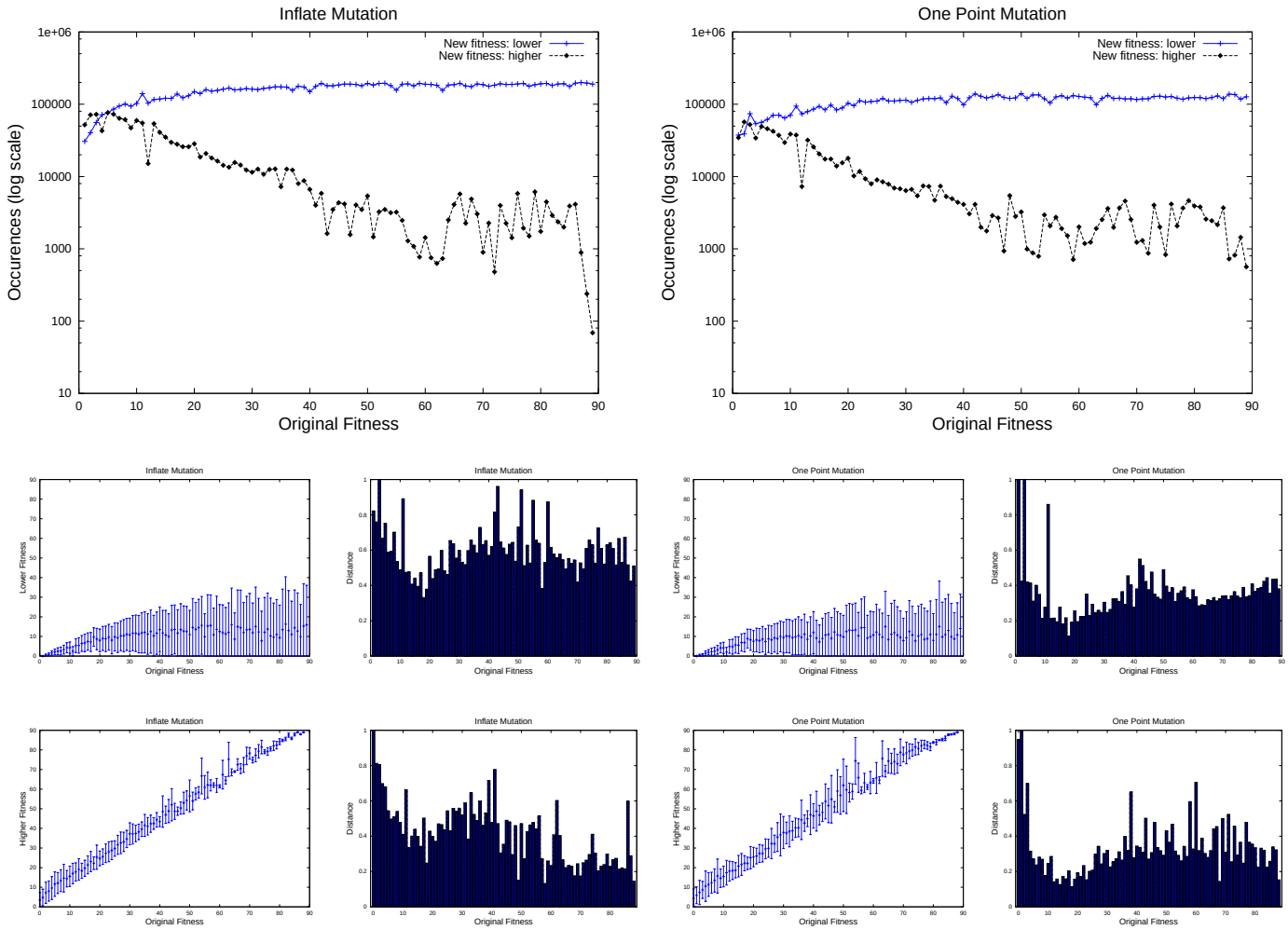

Fig. 2. Proportion of fitter and less fit offspring after applying inflate mutation mutation (top), less fit individuals after applying inflate mutation and their corresponding distance (centre) and fitter individuals after applying inflate mutation and their corresponding distance (bottom).

Fig. 3. Proportion of fitter and less fit offspring after applying one-point mutation mutation (top), less fit individuals after applying one-point mutation and their corresponding distance (top) and fitter individuals after applying one-point and their corresponding distance (bottom).

required to determine if the results presented here generalise to other problem domains, and we wish to conduct a similar analysis for the full suite of Genetic Programming search operators.

Acknowledgment

The authors would like to thank James McDermott for his useful comments on the paper. This publication has emanated from research conducted with the financial support of Science Foundation Ireland.

References

[1] M. Clergue, P. Collard, M. Tomassini, and L. Vanneschi. Fitness Distance Correlation and Problem Difficulty for Genetic Programming. In W. B. Langdon, E. Cantú-Paz, K. E. 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. K. Burke, and N. Jonoska, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2002*, pages 724–732, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[2] I. Dempsey, M. O'Neill, and A. Brabazon. *Foundations in Grammatical Evolution for Dynamic Environments*. Springer, 2009.

[3] E. Galván-López. *An Analysis of the Effects of Neutrality on Problem Hardness for Evolutionary Algorithms*. PhD thesis, School of Computer Science and Electronic Engineering, University of Essex, United Kingdom, 2009.

[4] E. Galván-López, S. Dignum, and R. Poli. The Effects of Constant Neutrality on Performance and Problem Hardness in GP. In M. ONeill, L. Vanneschi, S. Gustafson, A. I. E. Alcazar, I. D. Falco, A. D. Cioppa, and E. Tarantino, editors, *EuroGP 2008 - 11th European Conference on Genetic Programming*, volume 4971 of *LNCS*, pages 312–324, Napoli, Italy, 26–28 Mar. 2008. Springer.

[5] E. Galván López, R. Poli, and C. A. Coello Coello. Reusing Code in Genetic Programming. In M. Keijzer, U.-M. O'Reilly, S. 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.

[6] T. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico, Albuquerque, 1995.

[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, Berlin, 2002.

[9] J. F. Miller. An Empirical Study of the Efficiency of Learning Boolean Functions Using a Cartesian Genetic Approach. In W. Banzhaf, J. M. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. J. Jakiela, and

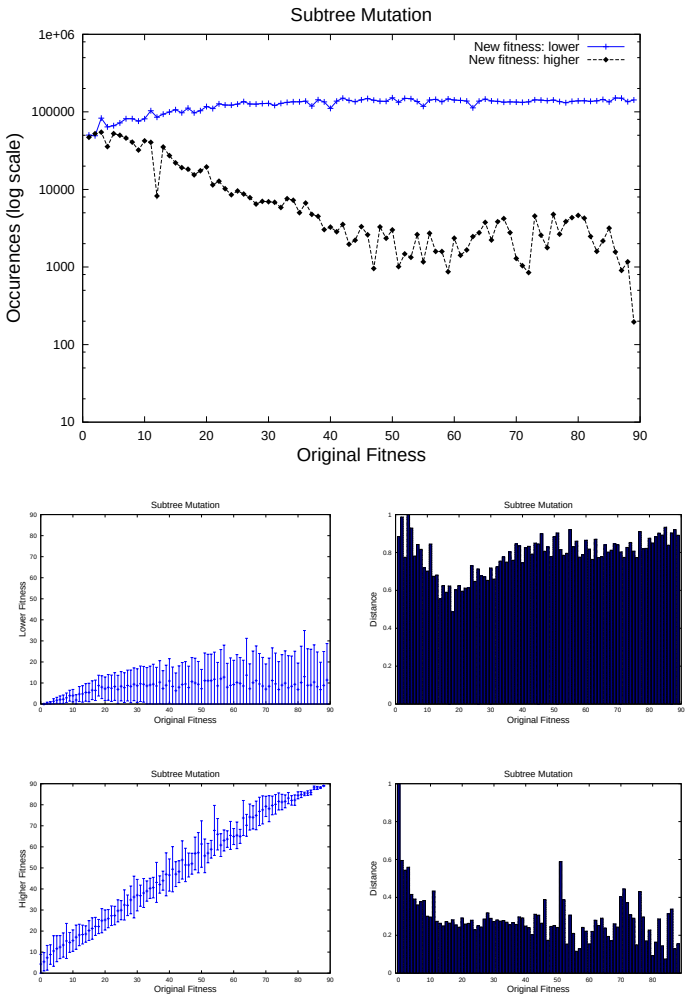

Fig. 4. Proportion of fitter and less fit offspring after applying subtree mutation mutation (top), less fit individuals after applying subtree mutation and their corresponding distance (top) and fitter individuals after applying subtree mutation and their corresponding distance (bottom).

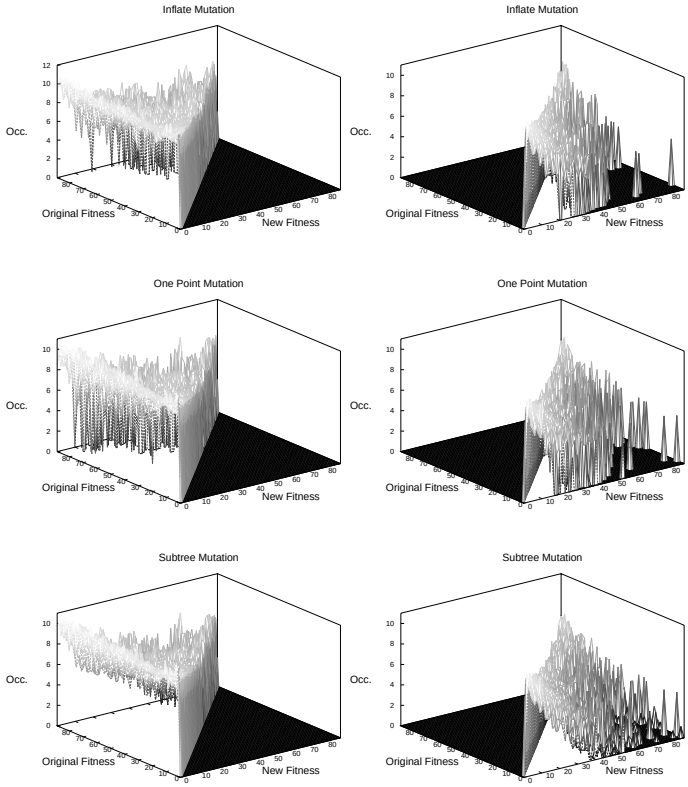

Fig. 5. Less fit offspring and occurrences (left) and fitter offspring and occurrences (right) when using inflate mutation (top), one-point mutation (middle) and subtree mutation (bottom) (note log z axis scale).

R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO'99*, volume 2, pages 1135–1142, Orlando, Florida, 13-17 July 1999. Morgan Kaufmann.

[10] M. O'Neill and C. Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers, 2003.

[11] R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[12] B. Punch, D. Zongker, and E. Godman. The Royal Tree Problem, A Benchmark for Single and Multi-population Genetic Programming. In P. Angeline and K. Kinnear, editors, *Advances in Genetic Programming 2*, pages 299–316, Cambridge, MA, 1996. The MIT Press.

[13] F. Rothlauf. *Representations for Genetic and Evolutionary Algorithms*. Springer, 2002.

[14] F. Rothlauf and D. Goldberg. Redundant Representations in Evolutionary Algorithms. *Evolutionary Computation*, 11(4):381–415, 2003.

[15] F. Rothlauf and M. Oetzel. On the Locality of Grammatical Evolution. In P. Collet, M. Tomassini, M. Ebner, S. Gustafson, and A. Ekart, editors, *EuroGP*, volume 3905 of *Lecture Notes in Computer Science*, pages 320–330. Springer, 2006.

[16] M. Tomassini, L. Vanneschi, P. Collard, and M. Clergue. A Study of Fitness Distance Correlation as a Difficulty Measure in Genetic Programming. *Evolutionary Computation*, 13(2):213–239, Summer 2005.

[17] L. Vanneschi. *Theory and Practice for Efficient Genetic Programming*. PhD thesis, Faculty of Science, University of Lausanne, Switzerland, 2004.

[18] S. Wright. The Roles of Mutation, Inbreeding, Crossbreeding and Selection in Evolution. In D. F. Jones, editor, *Proceedings of the Sixth International Congress on Genetics*, volume 1, pages 356–366, 1932.