A Fine-Grained View of GP Locality with Binary Decision Diagrams as Ant Phenotypes

James McDermott, Edgar Galván-Lopéz, and Michael O'Neill

Natural Computing Research & Applications group,
University College Dublin, Ireland
jamesmichaelmcdermott@gmail.com, edgar.galvan@ucd.ie, m.oneill@ucd.ie

Abstract. The property that neighbouring genotypes tend to map to neighbouring phenotypes, i.e. locality, is an important criterion in the study of problem difficulty. Locality is problematic in tree-based genetic programming (GP), since typically there is no explicit phenotype. Here, we define multiple phenotypes for the artificial ant problem, and use them to describe a novel fine-grained view of GP locality. This allows us to identify the mapping from an ant's behavioural phenotype to its concrete path as being inherently non-local, and show that therefore alternative genetic encodings and operators cannot make the problem easy. We relate this to the results of evolutionary runs.

Keywords: Genetic programming, phenotype, locality, problem difficulty, artificial ant.

1 Introduction

In evolutionary computation (EC), the *genotype* is the data structure, belonging to an individual, upon which the genetic operators work. It is mapped by some process to a *phenotype*, which in some sense represents the developed individual. Typically it is the phenotype which is evaluated for fitness, for example.

Some genetic programming (GP) variants have distinct genotypes and phenotypes, for example Cartesian GP [1], linear GP [2], grammatical evolution (GE) [3], and others. The case of standard, tree-structured GP is different. Depending on one's viewpoint, one might say that the genetic operators work directly on the phenotype, that the genotype-phenotype mapping is the identity map, or simply that no phenotype exists. Some methods of studying and comparing representations, common and useful in other areas of EC, turn out to be inapplicable to standard GP for this reason. One example is *locality*, an important measure of the behaviour of the genotype-phenotype mapping commonly used as an indicator of problem difficulty [4]. Some authors resort to measuring the locality of the genotype-fitness mapping as a substitute [5,6,7], but this does not tell the whole story. In particular, identifying a badly-behaved genotype-fitness mapping does not give us any clue about whether and how the mapping might be improved. Using phenotypes (e.g. [4]) can help to identify the components of the algorithm (e.g. a particular aspect of the encoding) responsible for the overall bad behaviour, or to conclude that no improvement is possible. In GP, it is possible to see program semantics or behaviour as a phenotype [8,9,10,11], and this is the approach taken here. GP locality has not been previously studied using such a definition of phenotypes.

Although the relatively poor performance of various GP techniques on the artificial ant benchmark cannot be regarded as an open problem [5], the study of locality can still teach us something new. The aims of this paper are thus to propose a fine-grained view of locality using two abstract, operator- and encoding-independent definitions of ant phenotypes; to use this model to study locality; and to relate the results with GP performance. Our methods may be applicable in a general way to other problems also.

The next section analyses previous work on locality and problem difficulty. In Sect. 3 we explain our view of locality, which allows for multiple phenotypes, and we give two distinct definitions for ant phenotypes. One of these is based on binary decision diagrams (BDDs), and an introduction to this topic is presented in Sect. 3.1. We present experiments on locality and evolutionary runs, and results, in Sect. 4; our conclusions are in Sect. 5.

2 Previous Work

Many authors have used landscape-oriented problem difficulty measures to try to predict search performance in EC. Examples include landscape correlation measures [12], epistasis [13], proportion of optima per program size [5], fitness-distance correlation (FDC) [14], FDC extensions including fitness clouds and negative slope coefficient [15,16], and locality and distance distortion [4].

In many of these cases, a proposed measure of difficulty (e.g. [14]) is followed by a counter-example (e.g. [17]): a problem which is easy to solve but is predicted to be difficult, or vice versa. In some cases the measure is then repaired or improved in some way (e.g. [16]), to be followed by new counter-examples (e.g. [18]). In addition to false predictions, there are problems of practicality. Jansen [19] summarised the situation for several such measures and demonstrated the key problems, including the requirement for very large sample sizes, unreliability of measures and counter-examples.

Despite this, research continues into methods of classifying problem difficulty, often with the aim of showing that one encoding or operator should be expected to improve performance relative to another. Here, we take the attitude that although no measure can predict performance perfectly, there are known results which can be explained in terms of problem difficulty, for example the poor performance of mutation-only GE relative to GP [20]. Such explanations contribute to an overall understanding of what makes EC work.

We concentrate on just one type of problem difficulty measure, *locality*, which in the general sense means the preservation of neighbourhood by a mapping. Rothlauf defines locality over the GA genotype-phenotype mapping and uses it to shed new light on several problems [4]. Although Rothlauf gives a numerical definition for the average locality of an encoding/mutation operator combination

[4] (p. 77), we will not use it in this paper since our aim is not to compare the locality of different encodings or operators. Instead, we will examine *the distribution of distances between neighbours after mapping* from one space to another. That is, we will study locality by taking pairs of neighbours, mapping each to a new space (i.e. genotypes to phenotypes, genotypes to fitness, or phenotypes to fitness), and looking at the distance between them in the new space.

3 A Fine-Grained View of Locality

In the abstract sense, locality refers to the preservation of neighbourhood by any mapping, not just the genotype-phenotype mapping common in many forms of EC. Therefore, when researchers (motivated by the absence of explicit phenotypes) study the preservation of neighbourhood by the genotype-fitness mapping in GP [5,6,7], this should be regarded as study of locality also. By defining explicit phenotypes in this paper, we aim to separate the genotype-fitness mapping into its component parts and study them separately.

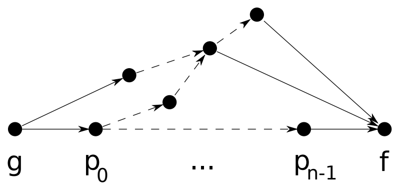


Fig. 1. Genotype g, phenotypes p_i, and fitness f. Arrows represent causality: for example, the calculation of p_0 depends on g, but g cannot be calculated from f.

Although it is common to think of each individual as having a single phenotype, a more general definition is possible: *any data structure which is calculated from the genome and which contributes to the calculation of fitness* may be seen as part of the "extended phenotype" [21]. The most general case is shown in Fig. 1. Each component of the mapping can, ideally, be studied separately.

In this study we focus on the artificial ant problem domain. Here, the problem is to find a program that can navigate a path of food laid out in cells on a grid [22] (pp. 147–155). The terminal set is `{move, right, left}`: these actions move the ant forward one square, and turn 90° to the right or left, respectively. Each consumes one time unit. The function set is `{iffoodahead, prog2, prog3}`. The first is a conditional: it executes its first argument if the ant perceives food directly ahead, and the second otherwise. The two remaining functions execute their two or three arguments in sequence. The most common grid layout is the *Santa Fe ant trail*, consisting of 89 food cells in a 32x32 grid, with the path characterised by twists and gaps. 600 time-steps are allowed.

The remainder of this section presents *two* definitions of ant phenotypes. Section 3.1 describes p_0, a phenotype based on binary decision diagrams, which represents the ant's *behaviour* in an encoding-independent way. Section 3.2 describes p_1, a cell-sequence phenotype, which represents the ant's *path* concretely. This leads to the overall model $g \rightarrow p_0 \rightarrow p_1 \rightarrow f$.

3.1 Binary Decision Diagram Phenotypes

One definition for ant phenotypes is suggested by [8,9]: an ant's behaviour is represented in an abstract form, inspired by the idea of stateful binary decision diagrams (BDDs) [23]. BDDs are a formalism for representing boolean functions. Any Boolean function composed of variables X_0, X_1, etc. and functions AND, OR, and NOT, for example, can be alternatively represented using a BDD.

Initially, a BDD may be thought of as a tree. At the root lives X_0, and it has two children each corresponding to X_1. At layer n live 2^n nodes corresponding to variable X_n. At the very bottom layer live nodes labelled 0 and 1. The essential idea is similar to that of a finite state machine. To evaluate a BDD, one traverses from the root, at each node choosing which of its two outgoing edges (labelled "high" and "low") to follow, depending on the value of the node's corresponding variable. The connectivity (i.e. the edges) ensures that the node one reaches at the end (0 or 1) is the value of the boolean function for the given variable values. In practice, it is common to use *reduced* BDDs, in which redundancies are eliminated: the BDD then no longer has a tree structure, since two divergent paths may re-join at a lower level.

Our definition of BDD-based ant phenotypes is similar but not identical to that of Beadle and Johnson [8,9]. The ant's behaviour is represented as a type of BDD: each node contains a sequence of zero or more action commands (left, right, and move), and each branch represents an if-statement. Branches rejoin after execution of an if-statement. In the ant problem there is only one "variable", the result of the `iffoodahead` predicate. This variable is *stateful*: it varies during an ant's run, so it is necessary to use multiple layers of nodes to represent behaviour. Since the ant's behaviour depends on the order in which it perceives cells, it is not possible to re-order the BDD (as for BDDs in other contexts) without altering behaviour. The mapping from genotype to BDD-phenotype is thus unambiguous. The algorithm for performing the mapping is omitted due to space constraints: code is available at `http://skynet.ie/~jmmcd/representations.html`. BDD-phenotypes are illustrated in Fig. 2.

We can equivalently write our BDD-phenotypes as strings. `L`, `R`, and `M` represent left, right, and move commands. A branch, conditional on the presence of food, is represented by an `<X,Y>` construct, where `X` and `Y` represent the two branches. We invert the standard BDD convention that the negative branch is written first, since in GP trees the positive branch of an if-statement is written first. The convention is arbitrary and the only effect of changing it is to make the BDDs easier to read.

Note that conversion from genotype to phenotype is not a simple matter of altering symbols. The sequencing implied by the `prog2` and `prog3` functions is abstracted away, and this allows some distinct genotypes to map to identical phenotypes. Several types of simplification are also required to obtain an abstract, canonical representation of ant behaviour:

– When one if-statement is nested directly inside another, one or other branch of the inner one will never be executed. That is, we can replace `<<X,Y>W,Z>` with `<XW,Z>`, and we can replace `<X,<Y,Z>W>` with `<X,ZW>` (here `W`, `X`, `Y` and `Z` are arbitrary sequences of actions).
– When the two branches of an if-statement end with the same action, it can be brought outside the branch. That is, we can replace `<XY,ZY>` with `<X,Z>Y`.
– As a result of the above simplifications, it may happen that an if-statement has two empty branches: `<,>` can be removed.

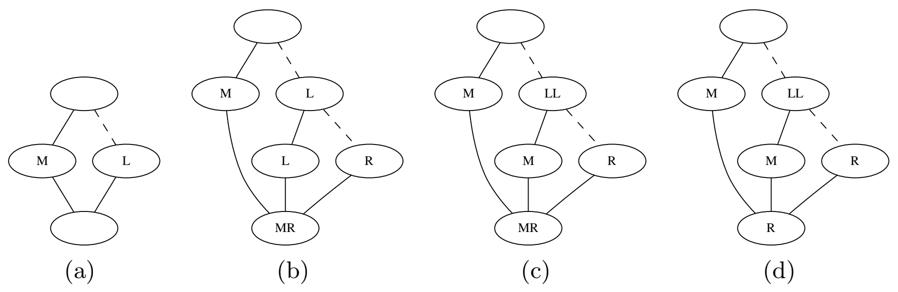


Fig. 2. BDD phenotypes. The positive and negative branches of an if-statement are drawn with solid and dashed lines respectively. Edges from non-branching nodes are drawn with solid lines. In (a) a simple example: the genotype is `(iffoodahead move left)` and the phenotype `<M,L>`. In (b) the genotype is `(prog3 (iffoodahead move (prog2 left (iffoodahead left (iffoodahead move right)))) move right)`. This translates to the phenotype `<M,L<L,<M,R>>>MR`. After removal of a redundant branch, we get `<M,L<L,R>>MR`, as shown. Phenotypic neighbours can have divergent fitness values: individual (c) has fitness 89, but its neighbour (d), created by a single phenotypic mutation, has fitness 1.

This representation can be run inside a suitable interpreter, and it will give the same ant path and fitness as the original GP genotype. Crucially, this representation is sufficiently abstract that it could also be used as a phenotype for several other types of GP in which the ant problem might be run, including GE, Cartesian GP, evolutionary programming (i.e. a finite state machine encoding), and others. The BDD phenotype representation also admits a (non-unique) backward mapping to genotype, not used in this paper.

We will measure distance between BDD-phenotypes using two string distance measures. *Normalised compression distance* has been previously used in diversity analysis [24]; *string edit distance* is well-known; both are general-purpose measures. This choice is made because no single distance measure of which we are aware is naturally suited for distances between BDD-phenotypes. Each measure gives at best an approximation to the "true" distance between a pair of phenotypes. However, improved measures must be left for future work.

We define BDD-phenotypic neighbourhood via "minimal edits" or mutations on phenotypes, which consist of insertion, deletion, or editing of any of the action commands `L`, `R` and `M`. Fig. 4 shows that phenotypic neighbours thus defined can have divergent fitness values. Our reasoning in considering only insertion, deletion and editing of the action commands is that non-minimal, structure-altering phenotypic edits will also lead *a fortiori* to divergent fitness values.

Examples of phenotypes and canonicalisation are shown in Fig. 2. We can illustrate the non-locality of the phenotype-fitness mapping: Fig. 2(c) shows the phenotype of an individual which solves the Santa Fe problem, i.e. has fitness 89, and Fig. 2(d) an individual created by a single phenotypic mutation which has fitness 1. The large change in fitness occurs because behaviour at each step depends on position and orientation after previous steps, and is repeated multiple times. Small changes in behaviour are thus multiplied.

3.2 Cell-Sequence Phenotypes

Another definition for an ant's phenotype is *the time-indexed sequence of cells it visits*, as illustrated in Fig. 3. This leads immediately to a natural definition of phenotypic distance: $d(a, b) = \sum_{t=0}^{T} d_c(a_t, b_t)$, where a_t and b_t are the position of ants a and b at time t, T is the maximum time, and d_c is a distance metric between cell positions, such as the toroidal taxi-driver's distance. The position of the food pellets mediates the behaviour of the ant but is not used in the calculation of these metrics. Note that a small change in cell phenotype will necessarily induce only a small change in fitness. The mapping from cell phenotype to fitness is, in other words, highly local by definition.

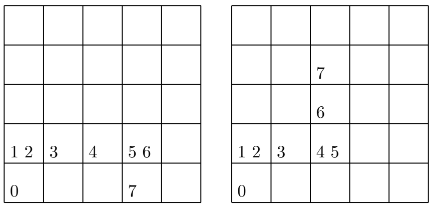


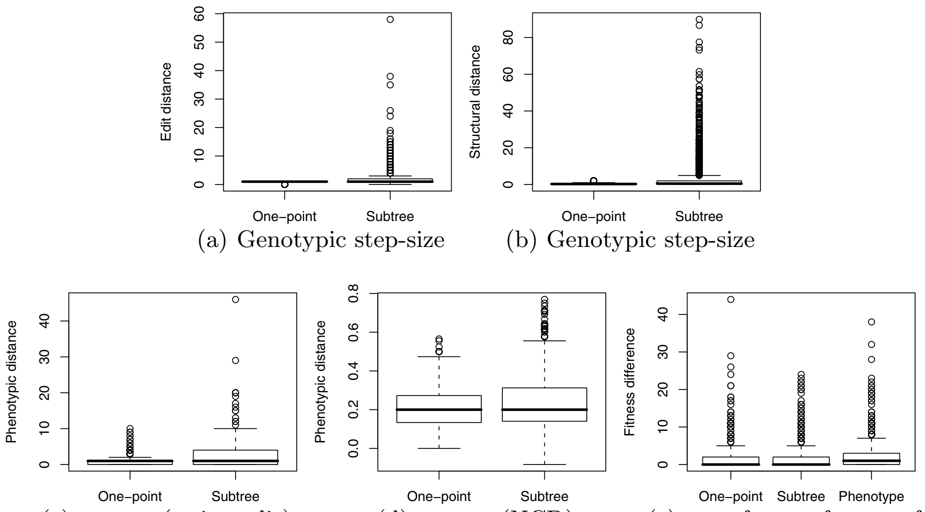
Fig. 3. Two ants' cell-sequence phenotypes. An integer t in a cell indicates that the ant was in that cell at time t. The food pellets are not shown. The distance d between these two ants is calculated as the sum of toroidal taxi-driver distances between corresponding points in the paths. Where the ants coincide (as for $t < 5$) the distance is 0. For $t = 5$ the distance is 1, for $t = 6$ it is 2, and for $t = 7$ it is 3 (take a shortcut through the bottom, emerging at the top), so the total distance is $0+0+0+0+0+1+2+3 = 6$.

4 Experiments and Results

Here we report the results of two experiments. The central hypothesis is that poor results in evolutionary runs can be explained in terms of locality measures.

We consider locality first, sampling individuals, mutating them using several methods, and measuring distances between pairs in several spaces. In every case, 1000 individuals were randomly generated, and a single mutation of the type shown (one-point, subtree, or phenotypic) was performed on each, yielding 1000 pairs of genotypic (respectively phenotypic) neighbours. The distance between pairs in the genotypic (respectively phenotypic, fitness) space was then recorded. Note that one-point mutation changes a single node per individual. Subtree mutation is a standard operator. Phenotypic mutation works as in Sect. 3.1.

Figs. 4(a) and 4(b) use two measures of genotypic distance (tree-edit distance and structural distance [25]) to show that different operators give different genotypic step-sizes. Fig. 4(c) shows that genotypic neighbours map to similar phenotypes when neighbourhood is defined by one-point mutation, but often do not when it is defined by subtree mutation. Fig. 4(e) (left and centre) shows that genotypic neighbours can have highly divergent fitness values, when genotypic neighbourhood is defined by either mutation operator. Finally, 4(e) (right) shows that BDD-phenotypic neighbours can have highly divergent fitness values also.



(a) Genotypic step-size (b) Genotypic step-size

(c) $g \rightarrow p_0$ (string-edit) (d) $g \rightarrow p_0$ (NCD) (e) $g \rightarrow f$, $g \rightarrow f$, $p_0 \rightarrow f$

Fig. 4. Different operators have different step-sizes ((a) and (b)). The genotype-to-BDD-phenotype mapping ((c) and (d)) can therefore be well- or badly-behaved, depending on the operator used to define neighbourhood. The genotype-to-fitness mapping ((e), left and centre) is badly-behaved for both. The BDD-phenotype-to-fitness mapping ((e), right) is badly-behaved.

The definition of non-trivial phenotypes allows us a fine-grained view of mapping behaviour. Recall that our model of the mapping is $g \rightarrow p_0 \rightarrow p_1 \rightarrow f$, where p_0 is the BDD-phenotype and p_1 is the cell-sequence. We already know that the overall map $g \rightarrow f$ is badly-behaved (confirmed by Fig. 4(e), left and centre): we can now seek to explain which of its components are responsible. The $p_1 \rightarrow f$ mapping has high locality by definition (see Sect. 3.2) and so is not to blame. However, $p_0 \rightarrow f$ has been shown to be badly-behaved (see Fig. 4(e), right, and recall Fig. 2(d)). Taking these results together shows that it is $p_0 \rightarrow p_1$, the mapping from the ant's abstract behaviour to its cell-sequence, which is badly behaved and at least partly responsible for the behaviour of the overall $g \rightarrow f$ mapping. Note, however, that subtree mutation can also cause bad behaviour in the $g \rightarrow p_0$ mapping (see Fig. 4(c)).

Table 1. Artificial Ant performance measured over 100 runs. Higher is better.

Algorithm/Setup	Mean Best Fitness	Std. Dev.	Hits
GP: no xover; subtree mut	60.54	9.98	6/100
GP: no xover; onepoint mut	51.24	7.22	0/100
GP: 9010 xover; subtree mut	61.27	10.10	5/100
GP: 9010 xover; onepoint mut	50.97	7.92	0/100
Random search:	60.86	13.93	1/100

In Table 1 we show the results of evolutionary runs using standard GP. The aim is to confirm that GP techniques perform poorly on the ant problem. 100 runs were performed with each setup, with population 500 and 50 generations. Typical parameters were used: 90/10 crossover probability 0.7, mutation probability 0.01 (but 1.0 for mutation-only GP), and maximum tree depth 7. A random search (implemented as a GP run of population 25,000 and 1 generation, with ramped half-and-half initialisation) is also reported for comparison. The only "knowledge" input to the random search was to avoid tree depths less than 4.

An ANOVA and pairwise t-tests were performed on the 100 best fitness values from each setup. Three set-ups (subtree mutation, crossover/subtree, and random search) each performed significantly better than the other two (one-point and crossover/one-point) ($p < 0.01$, Bonferroni correction for 10 pairwise t-tests). But the overall result is that GP performs poorly. There is little novelty in this: it reinforces the conclusion (in 1998) of Langdon and Poli [5], that the ant problem is difficult for many representations. Other representations including Cartesian GP and GE have since produced largely similar results [1,3].

It is noteworthy that subtree mutation does well relative to one-point, despite being more "randomising". It tends to take larger jumps in the search space. When a search space is badly-behaved, as here, highly local methods such as minimum-change operators lose any advantage they would have on smooth, well-behaved spaces. For difficult problems, then, random search tends to perform surprisingly well compared to more sophisticated algorithms.

5 Conclusions

We have proposed a fine-grained model of locality in the ant problem, in which the overall genotype-to-fitness map is broken up into three components with two intermediate phenotypes which have not been previously used in the study of locality. This new model allows us to identify the component—the map from the ant's abstract behaviour to its concrete path—responsible for the overall map's bad behaviour. We have performed various evolutionary runs and as expected we have added to Langdon and Poli's list of poor results [5] on this problem.

Our core conclusion is an attempt to explain these poor results: in the ant problem, the mapping from BDD-phenotypes to fitness is inherently badly-behaved. Since these BDD-phenotypes can function as encoding-independent behavioural phenotypes for many GP approaches to the problem, this result goes some way to explaining their universally poor performance. Thus, poor performance is not due to inadequate representations.

Although we have studied only the ant problem, the fine-grained model of locality proposed here may allow new insights into other problems also. Our definitions of phenotypes will not be directly usable in other problems, but possibilities are suggested by Beadle and Johnson's BDDs [8,9] and other semantic approaches. Other methods of characterising mapping behaviour, such as correlation analysis, might also benefit from similar fine-grained models.

Acknowledgments

Special thanks are due to Colin Johnson for an enlightening discussion of BDDs. JMcD is funded by the Irish Research Council for Science, Engineering and Technology under the Empower scheme. EG-L and MO'N thank the support of Science Foundation Ireland under Grant No. 08/IN.1/I1868.

References

1. Miller, J.F., Thomson, P.: Cartesian genetic programming. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 121–132. Springer, Heidelberg (2000)
2. Brameier, M., Banzhaf, W.: Linear genetic programming. Springer, New York (2006)
3. O'Neill, M., Ryan, C., Keijzer, M., Cattolico, M.: Crossover in grammatical evolution. Genetic Programming and Evolvable Machines 4(1), 67–93 (2003)
4. Rothlauf, F.: Representations for Genetic and Evolutionary Algorithms, 2nd edn. Physica-Verlag (2006)
5. Langdon, W.B., Poli, R.: Why ants are hard. In: Koza, J.R. (ed.) Proceedings of the Third Annual Conference on Genetic Programming, pp. 193–201. Morgan Kaufmann, Madison (1998)
6. Galván-Lopéz, E., McDermott, J., O'Neill, M., Brabazon, A.: Towards an understanding of locality in genetic programming. In: GECCO 2010: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, Portland, Oregon, USA. ACM Press, New York (2010)
7. Galván-Lopéz, E., McDermott, J., O'Neill, M., Brabazon, A.: Defining locality in genetic programming to predict performance. In: CEC 2010: Proceedings of the 12th Annual Congress on Evolutionary Computation, Barcelona, Spain. IEEE Press, Los Alamitos (2010)

8. Beadle, L., Johnson, C.G.: Semantic analysis of program initialisation in genetic programming. Genetic Programming and Evolvable Machines 10(3), 307–337 (2009)
9. Beadle, L., Johnson, C.G.: Semantically driven mutation in genetic programming. In: Proceedings of IEEE Congress on Evolutionary Computing, pp. 1336–1342 (2009)
10. Nguyen, Q.U., Nguyen, X.H., O'Neill, M.: Semantic aware crossover for genetic programming: The case for real-valued function regression. In: Proceedings of the 12th European Conference on Genetic Programming, pp. 292–302. Springer, Heidelberg (2009)
11. Jackson, D.: Phenotypic diversity in initial genetic programming populations. In: Esparcia-Alcázar, A.I., Ekárt, A., Silva, S., Dignum, S., Uyar, A.Ş. (eds.) EuroGP 2010. LNCS, vol. 6021, pp. 98–109. Springer, Heidelberg (2010)
12. Hordijk, W.: A measure of landscape. Evolutionary Computation 4(4), 335–360 (1996)
13. Altenberg, L.: NK fitness landscapes. In: Bäck, T., Fogel, D.B., Michalewicz, Z. (eds.) Handbook of Evolutionary Computation. IOP Publishing Ltd./Oxford University Press (1997)
14. Jones, T.: Evolutionary Algorithms, Fitness Landscapes and Search. PhD thesis, University of New Mexico, Albuquerque (1995)
15. Vanneschi, L., Tomassini, M., Collard, P., Verel, S., Pirola, Y., Mauri, G.: A comprehensive view of fitness landscapes with neutrality and fitness clouds. In: Ebner, M., O'Neill, M., Ekárt, A., Vanneschi, L., Esparcia-Alcázar, A.I. (eds.) EuroGP 2007. LNCS, vol. 4445, pp. 241–250. Springer, Heidelberg (2007)
16. Poli, R., Vanneschi, L.: Fitness-proportional negative slope coefficient as a hardness measure for genetic algorithms. In: Proceedings of GECCO 2007, London, UK, pp. 1335–1342 (2007)
17. Altenberg, L.: Fitness distance correlation analysis: An instructive counterexample. In: Proceedings of the Seventh International Conference on Genetic Algorithms, pp. 57–64 (1997)
18. Vanneschi, L., Valsecchi, A., Poli, R.: Limitations of the fitness-proportional negative slope coefficient as a difficulty measure. In: Proceedings of the 11th Annual conference on genetic and evolutionary computation, pp. 1877–1878. ACM, New York (2009)
19. Jansen, T.: On classifications of fitness functions. In: Theoretical aspects of evolutionary computing, pp. 371–386. Springer, Heidelberg (2001)
20. Rothlauf, F., Oetzel, M.: On the locality of grammatical evolution. In: Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A. (eds.) EuroGP 2006. LNCS, vol. 3905, pp. 320–330. Springer, Heidelberg (2006)
21. Dawkins, R.: The Extended Phenotype. Oxford University Press, Oxford (1982)
22. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. The MIT Press, Cambridge (1992)
23. Bryant, R.: Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Computing Surveys (CSUR) 24(3), 318 (1992)
24. Gomez, F.J.: Sustaining diversity using behavioral information distance. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation, Montréal, Canada, pp. 113–120. ACM, New York (2009)
25. Tomassini, M., Vanneschi, L., Collard, P., Clergue, M.: A study of fitness distance correlation as a difficulty measure in genetic programming. Evolutionary Computation 13(2), 213–239 (2005)