Chapter 4
How the Combinatorics of Neutral Spaces Leads Genetic Programming to Discover Simple Solutions

Wolfgang Banzhaf, Ting Hu, and Gabriela Ochoa

4.1 Introduction

A very general class of input-output systems has been found to have a very interesting property, conveyed by its features of being (i) discrete, (ii) computable, (iii) non-linear, together with some other smaller restrictions. Input-output systems of this type are found everywhere, from the molecular world of RNA, where the sequence-to-structure map can be conceptualized in those terms, via the maps of financial process models (like the Ornstein-Uhlenbeck financial model), to computational systems like L-systems for plant morphology or Boolean function maps as applied in simple models of evolutionary processes in Genetic Programming. More generally, many genotype-phenotype maps in Evolutionary Computation fulfill the conditions of such systems.

The property of interest is the fact that simpler outputs, i.e. patterns with less complexity are exponentially more abundant in such search spaces than more complex patterns. This result is, under certain conditions discussed in [5], independent of the particular mapping being applied. The genotype-phenotype map (G-P map, for short) of Genetic Programming systems can be seen a special case of this class of I/O systems.

W. Banzhaf (✉)
Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, USA
e-mail: banzhafw@msu.edu

T. Hu
School of Computing, Queen's University, Kingston, ON, Canada
e-mail: ting.hu@queensu.ca

G. Ochoa
Department of Computer Science, University of Stirling, Stirling, UK
e-mail: ochoa@cs.stir.ac.uk

S. Winkler et al. (eds.), *Genetic Programming Theory and Practice XX*,
Genetic and Evolutionary Computation,
https://doi.org/10.1007/978-981-99-8413-8_4

Here we use a simple Linear Genetic Programming (LGP) system working in a Boolean function space as a study tool to show that the combinatorial possibilities of neutral variations (modeled as neutral networks) provide the material by which the variation of abundances of phenotypes can be understood. This will allow us to better understand how individuals can move in such search spaces by the evolutionary process. We shall find that those phenotypes that have larger neutral networks will be more abundant and thus easier to find by an iterative stochastic process like evolution. In fact the relation between the dimension of the neutral subspaces and the abundance of phenotypes is exponential, providing a natural explanation for the above mentioned property.

The consequences of this property are wide-ranging. The realization that evolution in systems with neutral variations will be biased to find the simplest possible solutions to a problem first will allow to explain and predict that evolution in systems with neutrality will always prefer short-cuts, i.e. the simplest solutions, a fact that has caused surprise and consternation in the evolutionary computation community [15]. Second, the same property leads us to expect that evolutionary search will yield compact and most often simple, i.e. explainable, models in typical AI/ML applications such as pattern recognition or system modeling.

At the same time it helps to justify the use of repeated attempts (either independent in the form of multiple restarts or dependent in the form of populations) to search these spaces. As we shall show, the characteristics of neutrality will ultimately determine success or failure of a search algorithm. So what is indispensably required for such systems, is the existence of neutral variations, i.e. the fact that phenotypes can have different numbers of genotypes that map to them, thus providing the possibility of their variation in abundance.

In an earlier study [9] we examined the above mentioned LGP system and visualized the search process by looking at search trajectories in its network of solutions. This yielded a clear signal for the preference of such an evolutionary search process for pathways through the search space that lead via abundant intermediate stepping stone solutions. Here we shall extend this work by manipulating the degree of neutrality of phenotypes which requires us to change the connectivity in the neutral networks that comprise each of these phenotypes. We shall achieve this by making use of certain symmetries in the problem search space, as it refers to the logic equivalence of Boolean functions.

4.2 Related Work

4.2.1 I/O Systems

Dingle et al. report that discrete input-output systems of the form $f : I \rightarrow O$ which are governed by a mapping function f can be characterized using algorithmic information theory as having the interesting feature that the probability of finding an

output O, given a random input I, is determined by the Kolmogorov complexity of that output [5].

This is not true for all discrete input-output maps, but for a large general class of such maps, which include simple genotype-phenotype maps in Biology, e.g. the map from primary RNA sequences to their folding patterns [6], among others. In particular, the authors report that the probability $P(x)$ to find an output $x \in O$ is bounded by a quantity that depends exponentially on the Kolmogorov complexity of O:

$$P(x) \leq 2^{-(K(x|f,n)+\mathcal{O}(1)} \tag{4.1}$$

where $K(x|f,n)$ is the shortest program that produces x, given the computable input-output map $f : I \to O$ and the parameter n that characterizes the size of the input space. In other words, this probability is negatively exponentially related to the conditional Kolmogorov complexity of the output.

The authors of [5] further show that under not too onerous additional conditions this estimate becomes independent of the particulars of the map itself:

$$K(x|f,n) \approx K(x) + \mathcal{O}(1) \tag{4.2}$$

Those conditions apply in GP and are: (i) The map should have limited complexity, i.e. $K(f) + K(n) << K(x) + \mathcal{O}(1)$; (ii) the existence of redundancy, i.e. number of inputs N_I is much larger than number of outputs N_O, $N_I >> N_O$, which allows $P(x)$ to vary considerably, in principle, over the set of all outputs; (iii) the avoidance of finite size effects by requiring $N_O >> 1$; (iv) the nonlinearity of f, a feature fulfilled by many realistic maps.

4.2.2 RNA Studies

In earlier studies of the genotype-phenotype maps of RNA primary to secondary structure mappings it was found that many neutral networks exist which allow multiple RNA primary structures to map to few secondary structures [19]. The distribution of the number of primary structures to secondary structures was not uniform, but highly skewed: There were many secondary structures with just a few primary structures in their neutral networks and a small number of secondary structures with a large number of primary structures. This and other studies pointed to the importance of neutral networks and to the mechanisms by which they allow a search to succeed.

4.2.3 GP on Boolean Functions

Boolean functions were an early study object in Genetic Programming [13]. In [14] the fitness landscape of Boolean functions was systematically examined in the context

of a tree-based GP system and found to yield a curious feature—some of the Boolean functions were easy to find with small trees, while many were not found under such restrictions. If one allowed larger trees, the number of different Boolean functions increased, until at some point all of them could be found. The proportion of each Boolean function in the search space seemed to approach a certain asymptotic density. If one increased the number of allowed nodes in the tree, no further increase in the concentrations of a certain Boolean function could be found.

4.2.4 Neutral Networks

Neutrality itself has long been found to play a key role in the mechanisms of evolution. It was even stated that neutral evolution is the main engine of evolution [12]. Our own interests in neutral evolution go back to this inspiration from Biology [1]. While these ideas were formulated in a less formal system, the advent of the concept of neutral networks [22] brought a new perspective into these considerations which subsequently led to a study of neutral networks in Genetic Programming [2].

In [23] the authors examined some newly defined statistical measures of Boolean functions, in particular in connection with the neutrality features of many genotypes. The Boolean functions themselves were more complicated than in other work (multiplexers, parity problem), and the definition of neutrality made use of the notion of neutral networks.

Recently, Wright and Laue [25] examined the distribution of genotypes in a Cartesian Genetic Programming (CGP) representation, with a focus evolvability and complexity properties brought about by the genotype-phenotype maps in those systems.

4.2.5 Our Earlier Work

In our previous research, we used a Linear Genetic Programming (LGP) system, where a genotype is defined as a unique genetic program and a phenotype is defined as a Boolean relation a program represents, and constructed neutral networks to quantitatively study neutrality and related properties including robustness and evolvability in the system [7, 8, 10]. We reported a highly redundant G-P map and heterogeneous distribution of mutational connections among phenotypes.

Meanwhile, a recent graph-based model, search trajectory networks (STNs) [17, 18, 20], was developed to analyse and visualise search trajectories of any metaheuristics. Search trajectory networks are a data-driven, graph-based model of search dynamics where nodes represent a given state of the search process and edges represent search progression between consecutive states.

In our most recent work [9], we adopt STNs for an examination of the statistical behavior of searchers navigating the corresponding genotype space in Boolean LGP. Nodes are genotypes/phenotypes and edges represent their mutational transitions.

We also quantitatively measure the characteristics of phenotypes including their genotypic abundance (the requirement for neutrality) and Kolmogorov complexity. We connect these quantified metrics with search trajectory visualisations, and find that more complex phenotypes are under-represented by fewer genotypes and are harder for evolution to discover. Less complex phenotypes, on the other hand, are over-represented by genotypes, are easier to find, and frequently serve as stepping-stones for evolution.

4.3 Genotypes, Phenotypes, Behavior, Fitness

In order to have a clearer understanding of what is going on in evolution, we have to get a grasp on the search process and define terms carefully. Figure 4.1 shows a sketch of the search process at different levels of the (discrete) fitness in our Boolean function system. At each level of fitness we can imagine a neutral network (solutions with equal fitness) connected through edges symbolizing mutational (or variational) moves. There are connections between levels as well (moves that allow fitness to change), which usually go through *portal* nodes, nodes that provide connections between different fitness levels.

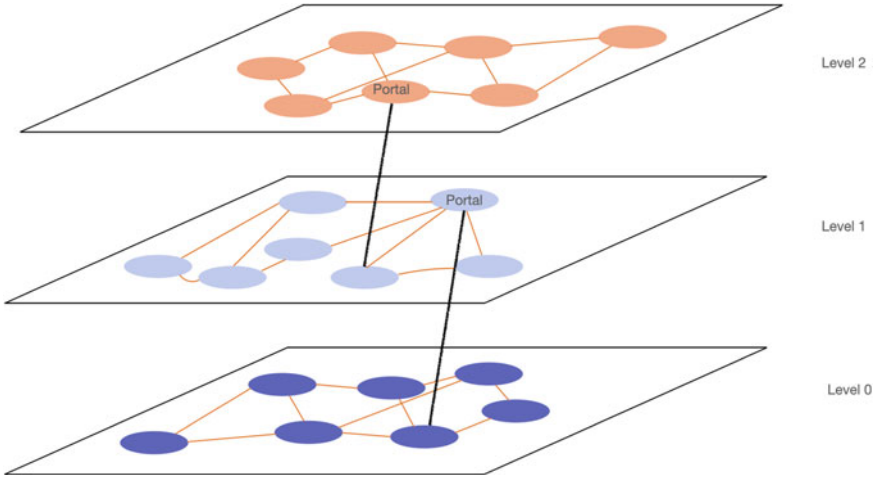

Fig. 4.1 Sketch of a network of neutral networks. Each level depicts one neutral network, with a discrete fitness value corresponding to its level. Nodes depict genotypes (genetic programs) which are connected within a level, reachable by neutral moves, with few nodes allowing jumps to a lower level (better fitness). The fitness of a node is measured by executing it and comparing the function it stands for with a target relation. The neutral networks are connected through what are called *portal* nodes to other neutral networks at a lower (better) fitness level

4.3.1 Discrimination of Genotypes and Phenotypes

The use of the term phenotype is somewhat ambiguous in Evolutionary Computation, extending a tradition from Biology. In Biology, an organism can have many different phenotypes. *Phenotype* is simply what is being observed, whether it is a structural trait of an organism, or its behavior. Here, we shall take a closer look and offer a more crisp definition.

Before we go there, let us start with the easiest definition—genotypes. A *genotype* is the pattern which is subject to the evolutionary operators of mutation and crossover, i.e. subject to genetic manipulation, see also Fig. 4.2. As a side: The reader might know Cartesian GP [16] which is frequently characterized as a graph GP system. However, we should emphasize that Cartesian GP with its linear sequence of numbers encoding graphs is actually a type of linear GP system.

Phenotypes are more difficult to define. There are different definitions of phenotypes, but generally, we understand *phenotypes* to be what is observed and subject to selection. The definition can be based on

1. its fitness;
2. its behavior; or
3. its effective structure.

What do we suggest adopting in the context of Genetic Programming? - We suggest discarding 1, since it is very clear in even this simple example of Boolean functions that different Boolean functions can have the same fitness, but that does not make them identical phenotypes! We have a choice of either 2 or 3. If we adopt 3, we are at the lowest level of resolution, but such a suggestion would run counter to what has been traditionally considered a phenotype in Biology. If we adopt 2, we are following at least part of that tradition. But we then need to discriminate further

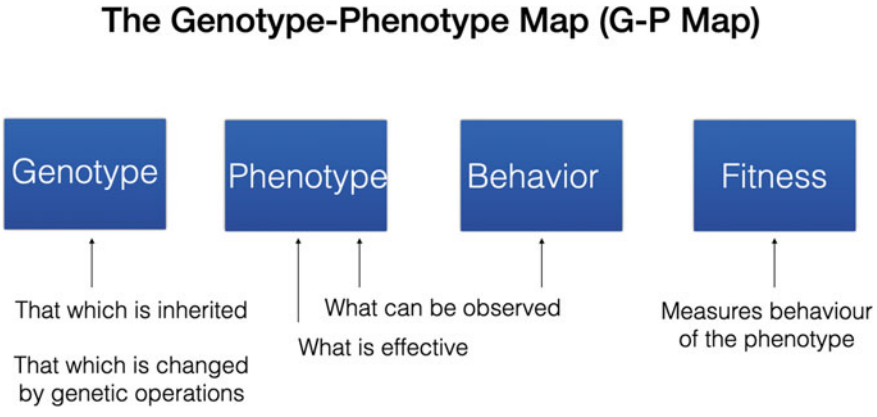

Fig. 4.2 The genotype maps to a phenotype which produces behavior that is judged by a fitness function

regarding the effective structure of the phenotype, since there are in GP many ways to produce a single behavior. Hence we suggest calling the different effective structures that produce the same behavior *isotypes*. Isotypes are semantically neutral, but different programs. Traditionally, they are considered the same phenotype, because they result in the same behavior.[1]

4.3.2 The Difference of Structural and Semantic Neutrality

This brings back a difference we made earlier between structural (or syntactic) and semantic neutrality/introns. Structural introns are parts of different genotypes that are structurally neutral variations, i.e. they don't affect the effective structure or isotype. Semantic introns, on the other hand, affect effective structure of a program, but not its behavior.

Given this situation, a remark is due on tree-based GP (TGP): Most (if not all) neutral variations in TGP runs are semantic, where most (if not all) neutral variations in LGP are structural. However, it is much easier to identify structural neutrality than it is to identify semantic neutrality. For the former, one only has to analyze a program once and this analysis is of order $O(L)$ with L being the length of the program. For semantic neutrality, one has to run the program and analyze the semantics for each node on k fitness cases, which for large k is possible only on a sample, and still time consuming.

Note that both GP representations can in principle have both types of neutrality. So, one would need to run both a structural and a semantic analysis on both representations to identify the full amount of neutrality in each individual. One would do the easier analysis first - for structural neutrality—then exclude those parts of the program from the second type of analysis. But the majority of reduction in complexity would come for LGP from the first step, while for TGP it would only come at the second step. That is why LGP is easier to handle, because one could just do the first step, and still end up with a good approximation of complexity, while for TGP the first step does not yield a good approximation.

If one wants to get a handle on the complexity of an evolved solution, it is important to identify the neutrality in a representation. What makes this difficult for tree GP is not only that semantic introns are time-consuming to identify, but also that genotype and phenotype in TGP are both trees, making it difficult to see the smaller (phenotypic) tree within the larger (genotypic) one. In other words, only a part of the GP tree genotype can count as the tree phenotype.

[1] An alternative is to divide phenotypes into a static (structural) and a dynamic (behavioral) part.

4.4 Methods

In this section, we describe the LGP system we used, the Boolean programs investigated in this study, and our visualization method using STNs.

4.4.1 Linear Genetic Programming

Linear Genetic Programming (LGP) [4] is a variant of GP [3] where a sequential representation of computer programs is employed to encode an evolutionary individual. Such a linear genetic program often consists of a set of imperative instructions to be executed sequentially. Registers are used to either read input variables (input registers) or to enable computational capacity (calculation register). One or more registers can be designated as the output register(s) such that the final stored value(s) after the program is executed will be the program's output.

4.4.2 Boolean Function Programs/Circuits

We use an LGP algorithm for a three-input, one-output Boolean function search application, similar to our previously examined LGP system [7, 10, 11]. Each instruction has one return, two operands and one Boolean operator. The operator set has four Boolean functions {AND, OR, NAND, NOR}, any of which can be selected as the operator for an instruction. Three registers $\mathtt{R}_1$, $\mathtt{R}_2$, and $\mathtt{R}_3$ receive the three Boolean inputs, and are write-protected in a linear genetic program. That is, they can only be used as an operand in an instruction. Registers $\mathtt{R}_0$ and $\mathtt{R}_4$ are calculation registers, and can be used as either a return or an operand. Register $\mathtt{R}_0$ is also the designated output register, and the Boolean value stored in $\mathtt{R}_0$ after a linear genetic program's execution will be the final output of the program. All calculation registers are initialized to FALSE before execution of a program. An example linear genetic program with three instructions is given as follows:

$$
\begin{aligned}
I_1 &: \mathtt{R}_4 = \mathtt{R}_2 \ \mathtt{AND} \ \mathtt{R}_3 \\
I_2 &: \mathtt{R}_0 = \mathtt{R}_1 \ \ \mathtt{OR} \ \ \mathtt{R}_4 \\
I_3 &: \mathtt{R}_0 = \mathtt{R}_3 \ \mathtt{AND} \ \mathtt{R}_0
\end{aligned}
$$

A linear genetic program can have any number of instructions; for the ease of sampling in this study, we use linear genetic programs that have a fixed length of 6 or 12 instructions.

The *genotype* in our GP algorithm is a unique linear genetic program. Since we have a finite set of registers and operators, as well as a fixed length for all programs, the genotype space is finite and we can calculate its size. For each instruction, two

registers can be chosen as return registers and any of the five registers can be used as one of two operands. Finally, an operator can be picked from the set of four possible Boolean functions. Thus, there are $2 \times 5 \times 5 \times 4 = 200$ unique instructions. Given the fixed length of six instructions for all linear genetic programs, we have a total number of $200^6 = 6.4 \times 10^{13}$ possible different programs. For 12 instructions, that number grows to 4×10^{27} different programs in the search space.

The *phenotype* in our GP algorithm is a Boolean relationship that maps three inputs to one output, represented by a linear genetic program, i.e., $f : \mathbf{B}^3 \rightarrow \mathbf{B}$, where $\mathbf{B} = \{\texttt{TRUE}, \texttt{FALSE}\}$. There are thus a total of $2^{2^3} = 256$ possible Boolean relationships. Having 6.4×10^{13} genotypes to encode 256 phenotypes, our LGP algorithm must have a highly redundant genotype-phenotype mapping. We define the *abundance/redundancy* of a phenotype as the total number of genotypes that map to it.

We choose the *fitness* of a linear genetic program as the deviation of the phenotype's behavior from a target Boolean function and want to minimize that deviation in the search process. Given three inputs, there are $2^3 = 8$ combinations of Boolean inputs. The Boolean relationship encoded by a linear genetic program can be seen as an 8-bit string representing the outputs that correspond to all 8 possible combinations of inputs. Formally, we define fitness as the Hamming distance of this 8-bit output and the target output. For instance, if the target relationship is $f(\texttt{R}_1, \texttt{R}_2, \texttt{R}_3) = \texttt{R}_1\ \texttt{AND}\ \texttt{R}_2\ \texttt{AND}\ \texttt{R}_3$, represented by the 8-bit output string of 00000001, the fitness of a program encoding the FALSE relationship, i.e., 00000000, is 1. Fitness is to be minimized and falls into the range between 0 and 8, where 0 is the perfect fitness and 8 is the worst.

4.4.3 Visualization Method

We use Search Trajectory Networks (STNs) [18], to analyze and visualize the behavior of the studied fitness functions and program lengths. STNs are a graph-based tool to study the dynamics of evolutionary algorithms and other metaheuristics. Originally, the model was used to track the trajectories of search algorithms in genotype space, where nodes represent visited solutions (or genotypes) during the search process, and edges represent consecutive transitions between visited genotypes. However, for large search spaces, this approach renders unmanageable models. Therefore, coarser models have been proposed where nodes represent sets of related genotypes rather than single ones. In particular, nodes can group genotypes expressing the same behavior or phenotype [9, 21].

This is the approach we follow here. We generate and visualize Phenotye Search Trajectory Networks, where the search space locations (nodes) are unique phenotypes, and edges represent consecutive transitions between phenotypes.

For constructing the Phenotype STN models, multiple adaptive walks (100 to be precise) are performed for each fitness function and program length. Adaptive walks are search trajectories in the fitness landscape that accept both neutral as well

as improving moves, with deteriorating fitness moves being prohibited. A move in our implementation is a single genotype mutation (change of a symbol for another selected uniformly at random). For each run, the sequence of genotypes visited is recorded. Thereafter, in a post-processing step, the visited genotypes across all runs are grouped into their corresponding phenotypes and the transitions between pairs of grouped nodes are also aggregated into single edges to construct a single graph model. Notice that some nodes and transitions may appear multiple times during the sampling process. However, the graph model retains as nodes each unique phenotype, and as edges each transition between pairs of visited phenotypes. Counters are maintained as attributes of the nodes and edges, indicating their sampling frequency. These counters are later used as visual attributes to depict the size of nodes and the widths of edges.

Once network models are constructed, we can proceed to visualize them and compute relevant network metrics. Node-edge diagrams are the most familiar form of network visualization, where nodes are assigned to points in the two-dimensional Euclidean space and edges connect adjacent nodes by lines or curves. Nodes and edges can be decorated with visual properties such as size, color and shape to highlight relevant characteristics.

Our STN visualizations use node colors to identify four types of nodes: (1) best nodes, which have the minimum possible fitness (zero) (2) neutral nodes, whose adjacent outgoing nodes have the same fitness, (3) portals, which link to a node with improved fitness, and (4) portals to best, which are portal nodes with a direct link to best solutions.

The shape of nodes identifies three positions in the search trajectories: (1) start of trajectories, (2) best node (fitness zero), (3) end of trajectories (which are not the best node), (4) intermediate locations in the trajectories.

Edges color indicate whether an edge is neutral or improving. Node sizes and edge thickness are proportional to their sampling frequency.

A key aspect of network visualization is the graph-layout, which accounts for the positions of nodes in the 2D Euclidean space. Graphs are mathematical objects, they do not have a unique visual representation. Many graph-layout algorithms have been proposed in the literature. Here, we use the layout we designed in [9] to visualize the phenotype STNs. This layout uses the fitness values as the nodes' y coordinates, while the x coordinates are placed as a horizontal grid according to the number of nodes per fitness level. This layout allows us to appreciate the search progression towards smaller (better) fitness values, as well as the amount of neutrality present in the search space.

4.5 The Role of Neutrality

In the literature of GP, there has been a discussion about the benefits of neutrality for a long time. It has often been said that neutrality harms, since it robs the evolutionary algorithm of a guidance toward the goal, as neutral networks have identical fitness for all its genotypes or phenotypes, depending on which level one studies.

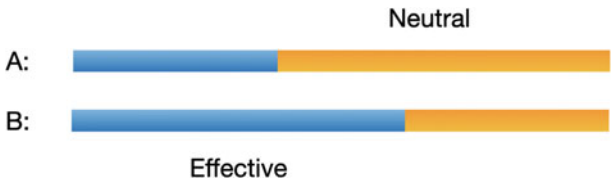

Fig. 4.3 The amount of neutrality of phenotype A, B depends on the space left for neutral variation (yellow part). The larger the neutral part, the more variants are available to that phenotype. At the same time, the shorter the effective part of the solution

However, it turns out that there are subtle differences between individuals, even if their fitness does not change on a neutral network. Two major differences are the abundance and the connectivity of phenotypes. As the results on discrete input-output maps discussed in the introduction show, some phenotypes are very frequently represented, compared to others, based on their complexity. Some are better connected in the network. The differences in abundance and connectivity guide a stochastic algorithm like evolution, even in the absence of fitness differences. This will lead to a higher probability of using highly abundant and better connected nodes in the network. These are two counter-acting tendencies: In a size limited search space (with a limit on the dimensionality of individuals - in our case 6 or 12 lines of code), abundance is tied to the amount of neutrality a phenotype gains via its neutral genotypes. If the total length of a program is fixed at L, the effective length is E and the neutral length is N, the following simple relation has to hold:

$$N = L - E \tag{4.3}$$

Figure 4.3 shows two examples. Note that the degree of neutrality depends exponentially on the number of neutral instructions. Thus a phenotype represented by genotype A will be more represented in the search space, and easier to find, compared to a phenotype represented by genotype B. It will be shorter in its effective part, thus less complex. This explains the phenomenon described earlier, of an negative exponential dependence of abundance on Kolmogorov complexity. On the other hand, a phenotype represented by genotype B has a better connectivity to other fitness because it has more effective instructions. This will make it easier to move via mutation from one phenotype to the other. These are counteracting tendencies that are able to explain why evolution in the longer run (when neutrality dominates) prefers low complexity solutions, while in the shorter run (when fitness effects dominate) it prefers to access solutions with higher complexity.

Given the role of neutrality in influencing search performance, a natural question to ask is whether and how we can increase the degree of beneficial neutrality in a search space. In the following two subsections, we shall discuss two ways of increasing neutrality that seem to have a positive effect on search efficacy.

4.5.1 Longer Programs

The simplest way to introduce more neutrality in the search space is to allow an increase in the length of programs, i.e. the number of instructions. While this does not change the ratio of abundances between phenotypes, at least beyond a certain minimum length (see [14]), it allows more connectivity between different phenotypes (i.e., non-neutral changes) through an expansion of the neutral networks of those different phenotypes. That is to say that the 1-mutant neighborhood of phenotypes includes more other phenotypes in larger networks than in smaller ones.

4.5.2 A New Fitness Function

The second method is to consider the fitness function to be used in the search space more carefully, according to the problem. If there is a symmetry in the problem, we might want to exploit that for the fitness function, allowing those phenotypes/solutions that show invariance under a certain transformation to be grouped together in the fitness function.

For example, in symbolic regression (SR), it is a well-known fact that a root mean square error (RMSE) fitness function which judges every data point separately, can be replaced by a correlation fitness function plus a post-processing step that uses the fact that all best fitness solutions under the correlation criterion are equally correct, modulo a linear scaling transformation at the end. This opens up multiple pathways to a good solution for the symbolic regression problem, where RMSE would only allow a very limited number.

4.5.2.1 A Failed Attempt

Here, we apply the same method in the context of Boolean functions. The equivalent of a symmetry (an invariance under certain transformations), is, for instance, the fact that a negated Boolean function is also a solution, provided one adds a negation gate at the output. As in the case of SR, it is only the local, relative difference of outputs between different input patterns for the function that are important. Thus, we define a new fitness function called *relative distance* (RD) between two bit strings $b^{(1)}$ and $b^{(2)}$ as follows:

$$RD(b^{(1)}, b^{(2)}) = \sum_{i=1}^{N-1} (b_i^{(1)} - b_{i+1}^{(2)}) \tag{4.4}$$

Minimizing RD will allow phenotypes that have the same local pattern change to be identically treated. If one reaches distance 0, a final decision has to be made for the first bit, which determines the bit pattern completely. Consider an example: Two 8-bit bitstrings $b_1 = 10100101$ and $b_2 = 01000010$ have a Hamming distance

of $HD = 6$. If looked at from the perspective of the relative distance, though, their distance is much smaller: $RD = 2$. This can be measured by transforming them into relative bit patterns $b_{1,r} = 1110111$ and $b_{2,r} = 110011$.

The phenotype of the solution is thus either the phenotype already found, or its negation. But during the search process prior to hitting fitness 0, both types of patterns are treated equally, which allows more pathways towards a solution. It should thus be easier to find solutions with the help of the RD fitness function than it is with the help of the Hamming distance (HD) fitness function (which is the equivalent of MSE in the binary space).

Unfortunately, that is not the case with this particular relative distance metric. While the additional symmetry of relative fitness might enhance the chances of finding the desired phenotype in a binary search space, this is balanced out by the countervailing influence of the enlarged search space. However, there is another symmetry that can serve to increase the neutrality of the search space: Applying a negation to the inputs of the Boolean function will produce three equivalent behaviors, those, together with the original behavior of the program, can be evaluated as the NegativeInputDistance fitness measure and minimized. This is what we are going to explore in the following in more detail.

4.5.2.2 The Negative Input Fitness Function

We are going to lump together the distances of 4 different phenotypic behaviors into one distance measure. For each of these behaviors we calculate Hamming distance d to the target t, then select the minimum as the NegativeInputDistance (ND):

$$ND(b, t) = \min_{i=1...4} d_i \tag{4.5}$$

where

$$d_i = \begin{cases} b & \text{for } i = 1 \\ d(\bar{b_i}, t) & \text{for } i = 2...4 \end{cases}$$

with $\bar{b_i}$ symbolizing a single negated input i. Once ND has converged to 0, we know we have hit the target, and it is only a matter of resolving which of the 4 possible phenotypes (lumped together in ND) produced the target behavior. This can be resolved in a post-processing step in constant time by testing which of the 4 possibilities produces the target phenotype behavior.

4.6 Results

4.6.1 A Comparison of Success Rates

Figure 4.4 shows the effects of longer programs and the new fitness function ND on the success rate of finding different phenotypes. Each of the 256 phenotypes is used as the target. A search starts with a randomly generated program and continues for 2,000 steps of point-mutations. 100 runs are collected for each experiment with a specified target phenotype. We plot the success rate, computed as the proportion of 100 runs that reach the target, as a function of the logarithm of the redundancy/abundance of the phenotype in the system.

We can see that redundancy is not equally distributed, but clustered around a number of redundancy values. This is a reflection of the discreteness of the Boolean search space used. We can discern around 10 different clusters, from very low redundancy (high difficulty) to very high redundancy (low difficulty). As shown in the figure, all 4 strategies are comparably successful when a target is very easy to reach (right side). For targets somewhat harder to discover strategies using the new fitness function ND are clearly more successful, even at the 100 % level. More difficult

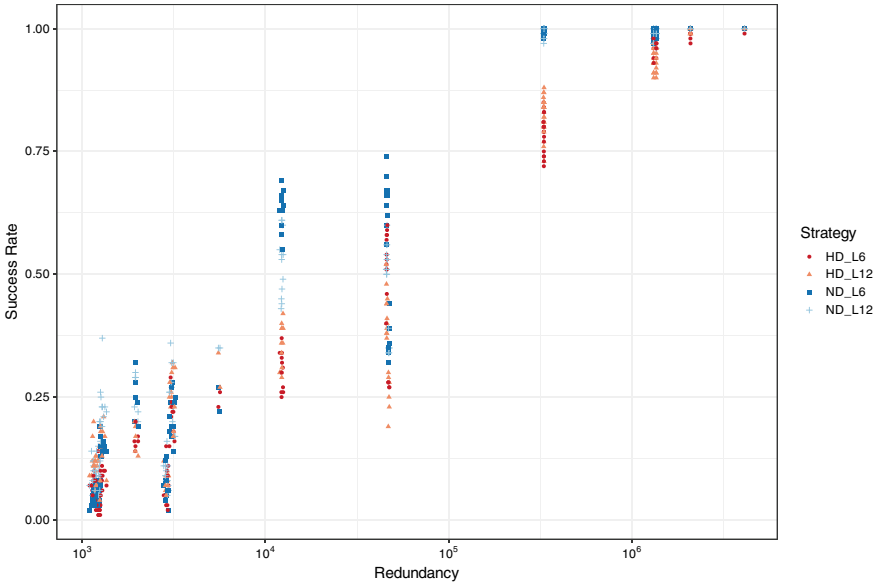

Fig. 4.4 Comparison of search strategies using fitness functions HD and ND, as well as program length 6 and 12. The 4 strategies are denoted using 4 different colors. Each data point represents one experiment using a specified phenotype as target. 256 distinct phenotypes are used as targets. Success rate is computed as the fraction of 100 runs in one experiment that reach a target phenotype within 2000 steps, and is plotted in relation to the target phenotype's redundancy. Phenotypes with higher redundancies are easier to find

phenotypes are found easier with the new ND fitness function, down to the most difficult phenotypes. For very difficult phenotypes, longer programs (L12) are more successful, with again ND fitness in the lead.

4.6.2 Comparison of Search Trajectory Networks for Three Targets

Next, we take a closer look at the search trajectories and investigate the effects of fitness ND and longer programs on neutrality. We pick three representative phenotypes as targets, phenotype 84 (redundancy 1,313,880), phenotype 140 (redundancy 331,241), and phenotype 215 (redundancy 3,060), with increasing difficulties to discover.

Figures 4.5, 4.6 and 4.7 show comparisons of STN visualizations for these three targets. Adapting the new fitness function ND and increasing program length allow more neutral moves (edged colored in grey), especially among stepping-stone phenotypes with medium fitness values. There are also more paths explored that lead to the target. These effects are increasingly prominent when searching for more difficult targets.

In addition to visualizing the STNs, we collect network metrics that quantitatively characterize these search networks. Figure 4.8 shows the comparison of these network metrics for the 4 search strategies. The number of nodes indicates distinct phenotypes explored. Neutral edges captures the proportion of neutral searches. While the differences are subtle, fitness ND and longer programs facilitate more thorough searches in the space through allowing more neutral moves. Degree best is the number of incoming edges to best phenotypes (targets and their equivalents). Strength best is the weighted sum of those incoming edges to best phenotype, given that edges are weighted based on their visit frequencies. As seen in the figure, more difficult targets have few paths directly connected to them. But fitness function ND and longer programs enable the discovery of more distinct paths finding the targets.

4.6.3 Simpler Solutions

It is a rather counter-intuitive fact that evolution seems to choose simpler solutions (phenotypes) over more complex ones. For many years our community has thought that evolution produces complex solutions. Bolstering this idea was the realization that above a certain complexity threshold there are many solutions to a problem, while below that threshold, there are none. In fact, there are normally only a few just above the threshold, and then there are many more as we increase the complexity. So it would seem that it is much easier to discover a complex solution than it would be to discover one of the few simple ones.

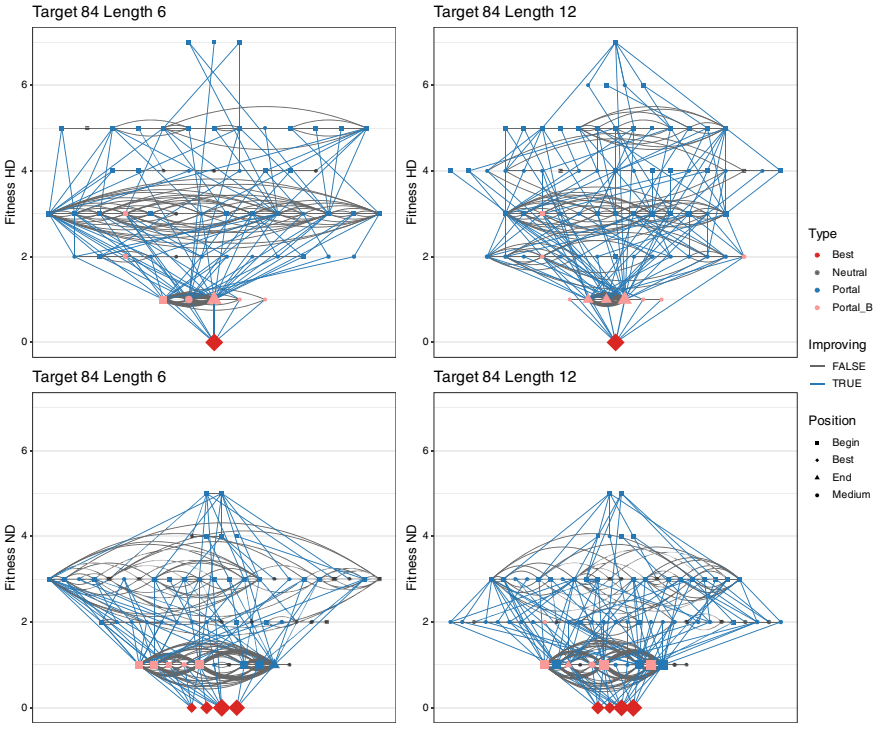

Fig. 4.5 Search trajectory network (STN) for easy target phenotype 84, comparing fitness function HD (top) and ND (bottom) and program length 6 (left) and 12 (right). Nodes are distinct phenotypes and edges show mutational transitions among phenotypes during searches. The figure shows the aggregation of search trajectories of 100 runs. Node shapes denote positions, i.e., beginning phenotype (randomly chosen), best phenotypes (fitness 0), end phenotype, and intermediate phenotype. Edge colors denote if a mutation improves fitness

But there is another influence at work in Genetic Programming—neutrality. Neutrality makes sure that simple solutions (in a limited search space) have many more neutral 'siblings' due to the combinatorics of the neutral space. As a result, the abundance of simple solutions is much higher and increases the chance of finding them.

Does that mean that evolution always finds the simplest solution? No, because it is not just the abundance of a phenotype that determines success, but also the connectivity of that phenotype. It is thus an effort (it requires optimization) to find the very simplest solution: Determining the Kolmogorov complexity of a bit string—in our case the phenotype's behavior—is an effort, but it can be achieved by a suitably set up GP search. We can therefore assume that evolution will generally find a simple solution, but not the simplest. In the normal case, the simplest solution might just not have enough connectivity in the network to be easily accessible, except in very

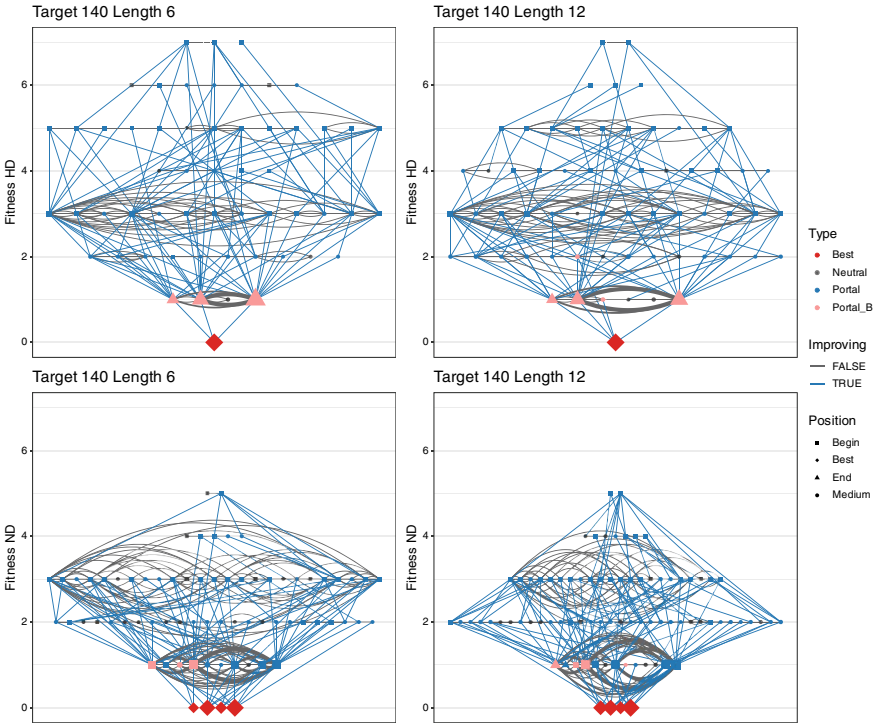

Fig. 4.6 Search trajectory network (STN) for medium target phenotype 140, comparing fitness function HD (top) and ND (bottom) and program length 6 (left) and 12 (right)

simple problems. That is because it needs to be accessed from a different fitness level, not from the neutral level on which it has so many siblings.

Here we consider the cumulative development of the complexity distribution over runs. Figure 4.9 shows the distribution of phenotypic complexity in 100 runs of target phenotype 140, L = 12, with high difficulty. We can see that complexity of programs at the beginning of runs is quite low: Most random programs clock in at complexity 1, 2 or 3. In fact, the median is 2. As the target phenotype with K-complexity 3 is reached, most solutions are beyond the minimum of 3, with the median of 5 at the time of discovery of the target. Beyond the initial discovery of the target, however, complexity of solutions trends downward, with a median of 4 at iterations 2000.

4.7 Discussion and Future Work

We have seen that evolution does something rather unexpected, if left to its own devices: It prefers simple over complex solutions, at least to the degree possible and achievable with reasonable effort. This is not a miracle but due to the higher

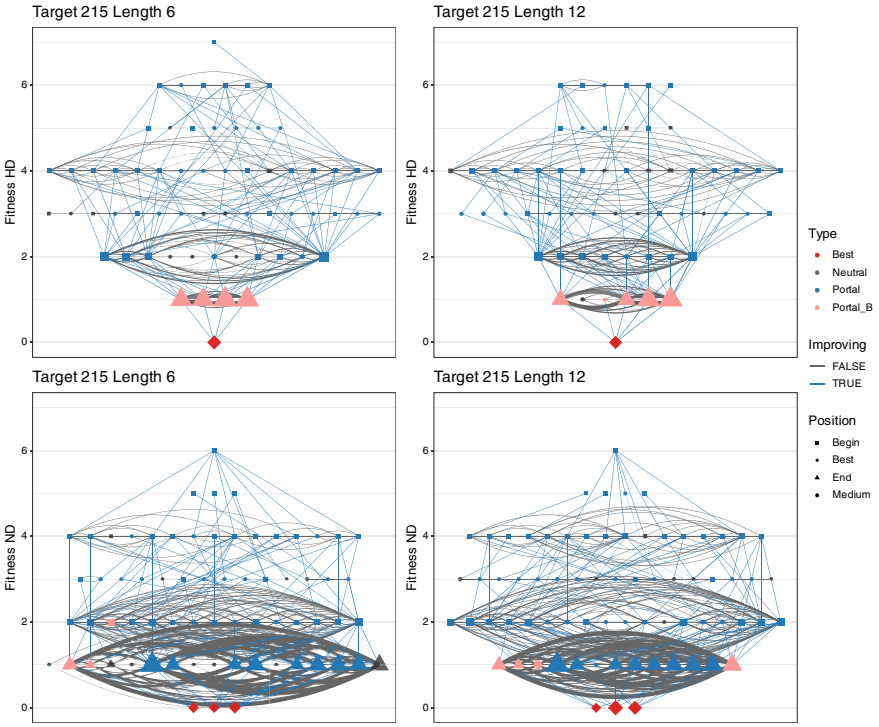

Fig. 4.7 Search trajectory network (STN) for hard target phenotype 215, comparing fitness function HD (top) and ND (bottom) and program length 6 (left) and 12 (right). The differences on the level(s) shortly before reaching the target are dramatic between HD and ND runs. A huge increase in possible pathways demonstrates the advantage of searching with the RD metric over the simple HD metric

abundance of simpler solutions over complex ones, at least in search spaces that are limited in size.

While this is an observation that could, in principle, be made in all GP systems allowing neutral solutions to play a role (i.e. those that do not exclude neutral search), it is particularly obvious in linear GP due to the ease of developing structural introns. While the same process is expected to go on in tree-based GP, it is less visible there due to the semantic nature of neutrality in that representation. Still it would be worthwhile to consider the same phenomenon in a TGP system. The corresponding intron discovery algorithm is easy to formulate, yet computationally expensive to execute.

The general result on input-output maps is based on algorithmic information theory. This should be expected to hold under many different circumstances, and it is no surprise that genotype-phenotype maps in GP fall under it. However, due to the nature of neutrality, a particularly intuitive explanation for the exponential distribution of abundance can be seen in our system: The increase in neutrality leads to exponentially more neutral solutions, due to the combinatorics allowed in

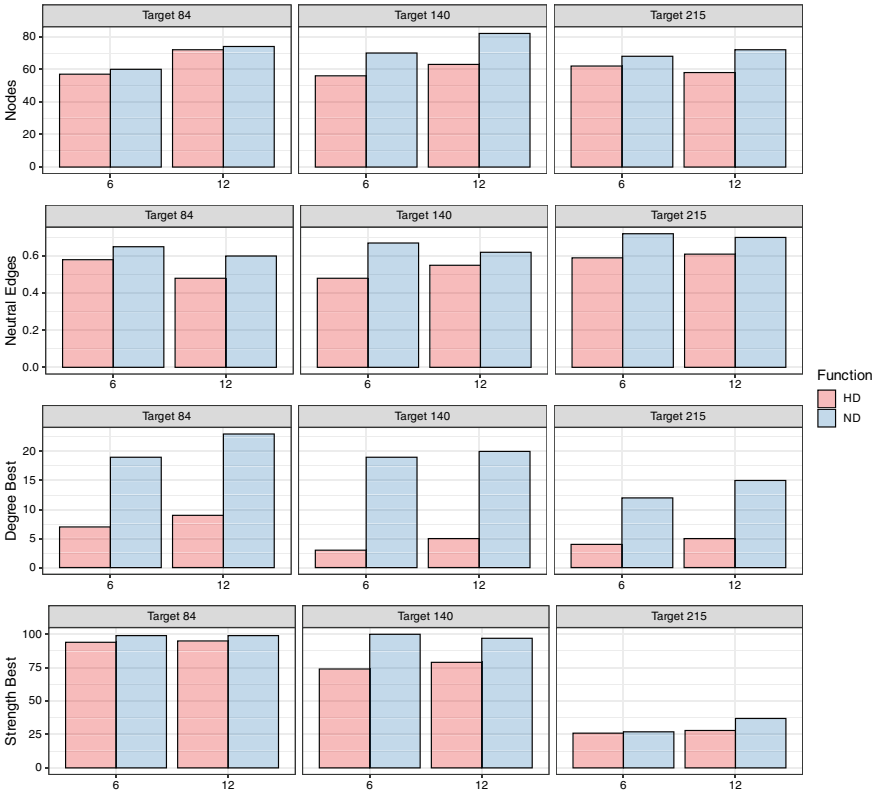

Fig. 4.8 Metrics of STNs comparing two fitness functions HD and ND and two program lengths 6 and 12. Number of nodes indicates how many distinct phenotypes are visited. The proportion of neutral edges shows edges in an STN that connect genotypes to phenotypes with the same fitness value. Degree best is the number of incoming edges to best phenotypes (fitness of 0). Strength best is the weighted degree (sum of edge weights) of best nodes

the neutral space. We think this is a key insight and it is essential not to over-engineer approaches to GP that do away with neutrality, or ignore it in favor of a presumed better efficiency of search algorithms by removing non-effective code during evolution. There is a reason for the emergence of neutrality from evolutionary processes, and it is counterproductive to fight this tendency. In fact it is not only an emergent process but it serves evolution at the same time to find simple solutions to problems faster than complex ones.

To leave the reader with something more heavy to ponder at the end: Does this tendency to prefer simple solutions perhaps shed some new light on the "unreasonable effectiveness of mathematics in the natural sciences" [24], a question Eugene Wigner has pondered for Physics, but as easily applicable to Biology?

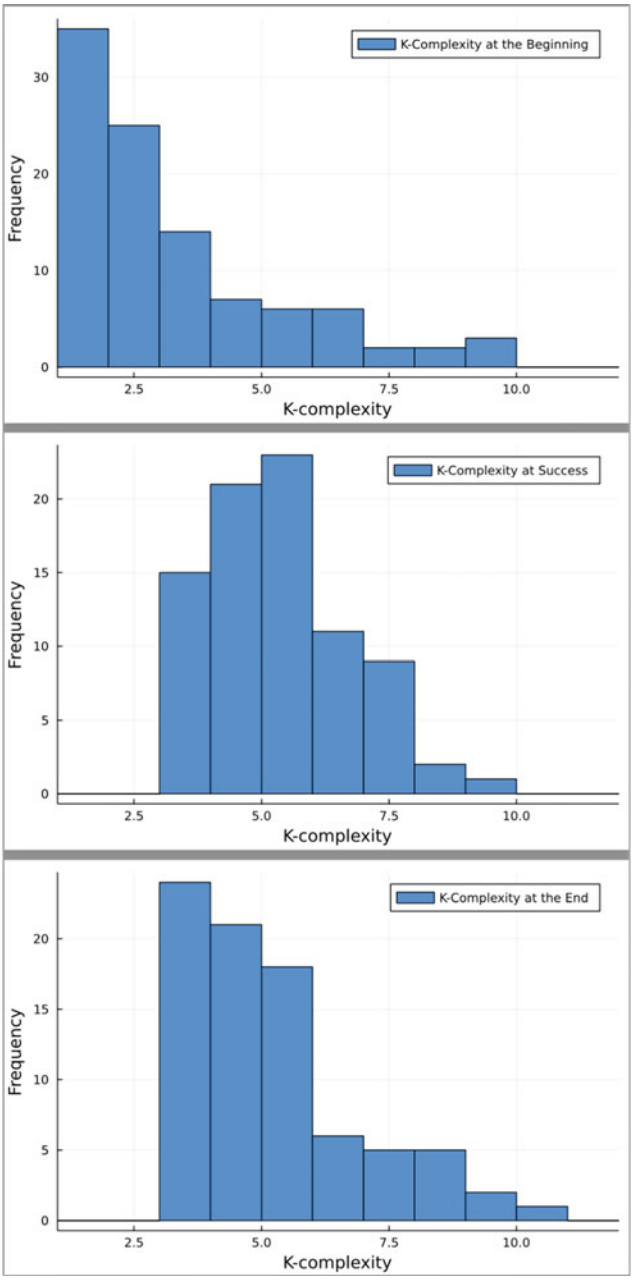

Fig. 4.9 Histogram of complexity distribution of phenotypes with target 140 at $L = 12$, HD fitness. Three cumulative distributions are shown, at the beginning of runs (random programs), at the moment they hit fitness 0, and at the end of 2000 iterations. The Kolmogorov complexity of the target phenotype is 3, thus no solution can be found with lower complexity. At the time of discovery, phenotypes are more complex than later

Acknowledgements The authors gratefully acknowledge the reviewers' insightful comments which helped to improve the manuscript.

References

1. Banzhaf, W.: Genotype-phenotype-mapping and neutral variation—a case study in genetic programming. In: International Conference on Parallel Problem Solving from Nature, pp. 322–332. Springer (1994)
2. Banzhaf, W., Leier, A.: Evolution on neutral networks in genetic programming. In: Genetic Programming—Theory and Practice III, pp. 207–221. Springer (2006)
3. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.: Genetic Programming—An Introduction. Morgan Kaufmann, Morgan Kaufmann Publishers 340 Pine Street, 6th Floor San Francisco, CA 94104 USA (1998)
4. Brameier, M., Banzhaf, W.: Linear Genetic Programming. Springer (2007)
5. Dingle, K., Camargo, C., Louis, A.: Input-output maps are strongly biased towards simple outputs. Nat. Commun. **9**, 761 (2018)
6. Dingle, K., Valle Perez, G., Louis, A.: Generic predictions of output probability based on complexities of inputs and outputs. Sci. Rep. **10**, 4415 (2020)
7. Hu, T., Banzhaf, W.: Neutrality and variability: two sides of evolvability in linear genetic programming. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, pp. 963–970 (2009)
8. Hu, T., Banzhaf, W., Moore, J.H.: The effect of recombination on phenotypic exploration and robustness in evolution. Artif. Life **20**(4), 457–470 (2014)
9. Hu, T., Ochoa, G., Banzhaf, W.: Phenotype search trajectory networks for linear genetic programming. In: Genetic Programming: 26th European Conference, EuroGP 2023, Held as Part of EvoStar 2023, Brno, Czech Republic, April 12–14, 2023, Proceedings, pp. 52–67. Springer (2023)
10. Hu, T., Payne, J.L., Banzhaf, W., Moore, J.H.: Robustness, evolvability, and accessibility in linear genetic programming. In: European Conference on Genetic Programming, pp. 13–24. Springer (2011)
11. Hu, T., Payne, J.L., Banzhaf, W., Moore, J.H.: Evolutionary dynamics on multiple scales: a quantitative analysis of the interplay between genotype, phenotype, and fitness in linear genetic programming. Gen. Program. Evol. Mach. **13**, 305–337 (2012)
12. Kimura, M.: The Neutral Theory of Molecular Evolution. Cambridge University Press, Cambridge, UK (1983)
13. Koza, J.R.: Genetic Programming. MIT Press, 12th floor of One Broadway, in Cambridge, MA 02142 (1992)
14. Langdon, W.B., Poli, R.: Foundations of genetic programming. Springer (2002)
15. Lehman, J., et al.: The surprising creativity of digital evolution: a collection of anecdotes from the evolutionary computation and artificial life research communities. Artif. Life **26**, 274–306 (2020)
16. Miller, J.F.: Cartesian genetic programming: its status and future. Gen. Program. Evol. Mach. **21**, 129–168 (2020)
17. Ochoa, G., Malan, K.M., Blum, C.: Search trajectory networks of population-based algorithms in continuous spaces. In: European Conference on Applications of Evolutionary Computation. EvoApps, pp. 70–85. Springer International Publishing, Cham (2020)
18. Ochoa, G., Malan, K.M., Blum, C.: Search trajectory networks: a tool for analysing and visualising the behaviour of metaheuristics. Appl. Soft Comput. **109**, 107,492 (2021)
19. Reidys, C., Stadler, P., Schuster, P.: Generic properties of combinatory maps: neutral networks of RNA secondary structures. Bull. Math. Biol. **59**, 339–397 (1997)

20. Sarti, S., Adair, J., Ochoa, G.: Neuroevolution trajectory networks of the behaviour space. In: European Conference on Applications of Evolutionary Computation, EvoApps, Lecture Notes in Computer Science, vol. 13224, pp. 685–703. Springer (2022). 10.1007/978-3-031-02462-7_43
21. Sarti, S., Adair, J., Ochoa, G.: Neuroevolution trajectory networks of the behaviour space. In: European Conference on Applications of Evolutionary Computation, EvoApps, Lecture Notes in Computer Science, vol. 13224, pp. 685–703. Springer (2022)
22. Schuster, P., Fontana, W., Stadler, P.F., Hofacker, I.L.: From sequences to shapes and back: a case study in RNA secondary structures. In: Proceedings of the Royal Society of London. Series B: Biological Sciences, vol. 255(1344), pp. 279–284 (1994)
23. Vanneschi, L., Pirola, Y., Mauri, G., Tomassini, M., Collard, P., Verel, S.: A study of the neutrality of boolean function landscapes in genetic programming. Theor. Comput. Sci. **425**, 34–57 (2012)
24. Wigner, E.P.: The unreasonable effectiveness of mathematics in the natural sciences. Commun. Pure Appl. Math. **13**, 1–14 (1960)
25. Wright, A.H., Laue, C.L.: Evolvability and complexity properties of the digital circuit genotype-phenotype map. In: Proceedings of the Genetic and Evolutionary Computation Conference—GECCO 2021, pp. 840–848. ACM Press (2021)