Enhancing Cartesian genetic programming through preferential selection of larger solutions

Nicola Milano1 · Stefano Nolfi1

Received: 21 June 2019 / Revised: 23 March 2020 / Accepted: 3 May 2020 / Published online: 17 May 2020
© Springer-Verlag GmbH Germany, part of Springer Nature 2020

Abstract
We demonstrate how the efficiency of Cartesian genetic programming methods can be enhanced through the preferential selection of phenotypically larger solutions among equally good solutions. The advantage is demonstrated in two qualitatively different problems: the eight-bit parity problems and the “Paige” regression problem. In both cases, the preferential selection of larger solutions provides an advantage in terms of performance and of speed, i.e. number of evaluations required to evolve optimal or high-quality solutions. Performance can be further enhanced by self-adapting the mutation rate through the one-fifth success rule. Finally, we demonstrate that, for problems like the Paige regression in which neutrality plays a smaller role, performance can be further improved by preferentially selecting larger solutions also among candidates with similar fitness.

Keywords Evolvability · Robustness · Cartesian genetic programming · Boolean circuits · Regression problem · Even-parity problem

1 Introduction
Evolutionary algorithms are known for their ability to discover simple solutions, i.e. the tendency to select the smallest available solutions. This is not surprising since evolution typically proceeds by selecting minimal sub-optimal solutions first and by then progressively extending those solutions until, possibly, an optimal solution is found. From an application point of view, the tendency to select compact solution is also desirable since it permits to generate solutions that are cheaper, lighter, and/or faster than non-compact solutions.

Compact solutions, however, can be less evolvable than more elaborated solutions. Consequently, the tendency to select minimal solutions might reduce the chance to produce behavioral variation as a result of genetic variations—a property that constitutes a crucial prerequisite for evolutionary progress.

In this paper we verify whether the preferential selection of phenotypically large solutions permits to preserve and/or to enhance the evolvability of candidate solutions and consequently the probability to evolve high performing solutions in the context of Cartesian genetic programming. The results collected on two qualitatively different problems: the eight-bit parity problem and the Paige regression problem indicate that the preferential selection of phenotypically larger solutions speed-ups the evolutionary process and permits to discover better solutions. We show that the advantage of the preferential selection of larger solutions can be extended by self-adapting the mutation rate through the one-fifth success rule. Finally, we show that for problems in which neutrality plays a smaller role, the advantage of the preferential selection of larger solutions can be extended by preferring larger solutions also among solutions displaying similar performance.

The paper is organized as follow. In Sect. 2 we briefly review Cartesian genetic programming. In Sect. 3 we illustrate the contribution of this work to the state of the art. In Sects. 4 and 5 we describe the experimental method and the obtained results. Finally, in Sect. 6 we draw our conclusions.
2 Cartesian genetic programming

Cartesian genetic programming (CGP, [16, 18]) is a form of Genetic Programming [12, 13]. It is usually used to evolve acyclic computational structures of nodes (graph) indexed by their Cartesian coordinates but can also be extended to evolve recurrent (cyclic) structures [30]. The method has been successfully applied to evolve digital circuits [19, 20], robots’ controllers [6], Atari games players [35], neural networks [10], image classifier [7], molecular docking [2] and regression programs [8].

CGP uses a vector of integer numbers to encode a graph constituted by nodes and connections. The properties of each node are encoded in a tuple of integers (genes) that encodes the function of the node, chosen from a list of available functions, and the indexes of the input of the nodes. Nodes can take input from either the problem inputs or from any node preceding them in the chromosome. A final set of tuples including a single integer specify the index of the nodes that are used to produce the output of the circuit (Fig. 1).

The initial chromosome is generated randomly. More specifically, the value of the genes encoding the function of each node are generated randomly with a uniform distribution in the range [0, NGF] where NGF is the number of alternative functions included in the functions’ list. The value of the genes encoding the input of a node are generated randomly with a uniform distribution in the range [0, G−1], where G is the index of the corresponding node. Offspring are generated by creating a mutated copy of the parent chromosome. Mutations are realized by replacing a fraction of randomly selected genes with a new integer selected randomly with a uniform distribution in the ranges described above. Nodes can eventually be arranged in pre-determined layers by limiting the range of the input indexes to the nodes contained in the previous layer.

Usually only a small fraction of the nodes participates to the generation of the output values. Indeed, although each node has inputs, the output of a node does not necessarily have to be used by the inputs of later nodes that projected their output to the output of the circuit either directly or indirectly. The nodes with un-used outputs are non-functional since they do not influence the overall output of the program. Similarly, genes encoding nodes with un-used outputs are non-functional since they do not affect the performance of the program (functional and non-functional nodes and genes are indicated in black and gray, respectively, in Fig. 1). Non-functional nodes can be eliminated from a program without altering the behavior of the program. However, non-functional nodes and genes that are non-functional in a circuit can become functional in the offspring of that circuit as a result of mutations affecting “downstream” genes. Similarly, formerly functional genes can become non-functional as a result of mutations affecting downstream genes [17].

The definition of non-functional nodes can be extended to include also the nodes that project connections to the output nodes. The last tuple encodes the index of the node that is used as output. The functional and non-functional genes/nodes are shown in black and grey, respectively (see text). Indexes are constituted by integer numbers and are used to identify the inputs of the circuit, the nodes, and the output of the circuit.

(A) [(0,1,2),(1,0,3),(2,3,1),(5)]

Fig. 1 Example of a chromosome and of a corresponding CGP program with three inputs, three nodes, and one output. The three portions of the figure illustrate the chromosome (a), the circuit (b) and the list of available node type (c). The first three tuples of the chromosome encode the function and the input of the three corresponding nodes. The last tuple encodes the index of the node that is used as output. The functional and non-functional genes/nodes are shown in black and grey, respectively (see text). Indexes are constituted by integer numbers and are used to identify the inputs of the circuit, the nodes, and the output of the circuit.
nodes indirectly but that do not alter the state of the output. We restrict our analysis to the former type of non-functional node only (Fig. 1).

CGP programs are usually evolved through a \((\mu + \lambda)\) Evolutionary Strategy [24, 25], where \(\mu\) is set to 1 and \(\lambda\) is usually set to 4 [16, 18], \((1 + \lambda)\) Evolutionary Strategies can also be referred as hill climbers. This means that during each generation a single parent produces four offspring with mutations, as described above. The mutation probability is usually uniform for all genes. When none of the offspring outperform the parent but at least an offspring equals the parent, the offspring is selected. The comparative study reported in Milano et al. [14] indicates that the utilization of a single parent might play an important role in CGP. Indeed, CPG programs evolved on the five-bits parity task with \(\mu = 1\) largely outperformed programs evolved with \(\mu > 1\).

### 3 Contribution to the state of the art

Only few studies investigated the relation between the size of the evolving solutions and evolvability defined as the propensity to generate high-performing solutions.

Several works investigated the possibility to minimize the size of the evolving solutions. The objective of these works, however, is to identify the most compact solutions among the set of equally good solutions, e.g. the circuits that can be manufactured by using the smaller number of electronic components. This is typically realized by preferentially selecting phenotypically smaller solutions once an optimal solution is found [3, 4, 11, 32]. The objective of the experiments reported in this paper, is different. We investigate whether the preferential selection of larger solutions lead to the discovery of solutions that outperform those obtained without preferential selection.

The presence of a positive correlation between size and evolvability was reported by Raman and Wagner [23]. The authors arrived to this conclusion by measuring the phenotypic variability of randomly generated circuits and by finding a positive correlation between phenotypic variability and size of the circuit. Phenotypic variability is defined as the propensity of circuits to generate offspring circuits performing different functions as a result of random mutations. The conclusion thus is based on the hypothesis that the higher the phenotypic variability of circuits is, the higher the chance that evolution will discover better solutions as a result of variation and selection is. Since phenotypic variability correlates positively with the size of the circuits, the evolution of larger circuits should lead to better solutions than the evolution of smaller circuits.

Such hypothesis was confirmed by the analysis performed by Milano et al. [14] on circuits evolved for the ability to solve a five-bit even parity problem. Indeed, the authors observed a strong positive correlation between the performance and the size of the evolving circuits. In addition, this study demonstrated that the characteristics of the evolutionary algorithm can influence the size of the evolving circuits which, in turn, influences the performance of the algorithm. In particular they claimed that the fact that \((1 + \lambda)\) ES largely outperform \((\mu + \mu)\) ES can be explained by the fact that the latter algorithm selects smaller circuits than the former. This is due to the fact that offspring of smaller circuits are more robust to mutations than offspring of larger circuits. Consequently, offspring of smaller circuit have a greater chance to be retained in future generations than offspring of smaller circuits. This effect does not manifest in the \((1 + \lambda)\) ES method in which offspring are generated from a single parent.

The relation between robustness to mutations and phenotypic variability can be illustrated by relying on the notion of neutral network [27, 31]. A neutral network is constituted by a series of nodes connected through bidirectional links in which nodes correspond to candidate solutions with identical fitness and links correspond to single mutations that enable to transform a candidate solution in genetically different candidate solution achieving the same fitness. The number of links per nodes and consequently the robustness to mutation is higher in the central part of the neutral network [27, 31]. During neutral evolution, the movement of the population over the neutral network is not entirely random but is directed preferentially toward high connected parts of the network which include phenotypes robust to mutations [31]. Such robust regions of the neutral networks include minimal solutions that achieve robustness by minimizing the number of functional genes but which have a low phenotypic variability [9, 33, 34]. The preferential selection of larger solutions drives the evolutionary search toward the peripheral areas of the neutral network and/or toward solutions that achieve robustness through redundancy or degeneracy [1, 15, 28]. These areas are characterized by a greater phenotypical variability than areas including minimal solutions. In other words, the preferential selection of larger solutions drives the evolutionary search toward solutions that are more evolvable.

In this paper we demonstrate for the first time that the preferential selection of larger circuits among equally good candidate solutions enables to synthesize better solutions, i.e. enable to generate circuits that outperform the circuits evolved in control experiments without such preferential selection.

Moreover, we demonstrate that a preferential selection of larger circuits among similar performing circuits permits to generate better solutions in continuous problems in which the chances to choose circuits among equally good candidates is much smaller.
4 Method

As mentioned above, we compared the performance of two algorithms: a standard \((1 + \lambda)\) ES and a \((1 + \lambda)\) ES-PL algorithm that preferentially select programs with functionally larger phenotypes. In both cases, during each generation a single parent produces \(\lambda\) offspring with mutations. In the case of the \((1 + \lambda)\) ES method, the \(1 + \lambda\) candidate solutions are sorted primarily on the basis of their fitness (in descending order), and secondarily on the basis of whether they correspond to an offspring or to the parent (in this order). The first individual is then used to replace the parent. In the case of the \((1 + \lambda)\) ES-PL method, instead, the \(1 + \lambda\) individuals are sorted primarily on the basis of their fitness (in descending order), secondarily on the basis of the size of their functional circuit (in descending order), and then on the basis of whether they correspond to an offspring or to the parent (in this order). The first individual is then used to replace the parent. We did not skip the evaluation of identical individual [5] since the probably to generate them is low in our experimental setting. Consequently, the computation cost for identifying identical genotypes would exceed the computational cost of skipped evaluations.

The chromosomes of candidate solutions include \(N\) three-digit tuples that encode the function and the input of \(N\) corresponding nodes and \(M\) one-digit tuples that encode the index of \(M\) corresponding output nodes. The value of the genes encoding the input of a node are generated randomly with a uniform distribution in the range \([0, G–1]\), where \(G\) is the index of the corresponding node. Offspring are generated by creating a mutated copy of the parent chromosome. The genes of the initial chromosome are created randomly with a uniform distribution in the range described above. Mutations are realized by replacing a fraction of randomly selected genes with new integers selected randomly with a uniform distribution in the ranges described above. Nodes are not arranged in fixed layers (i.e. we used a 1D representation).

In the first series of experiments, programs were evolved for the ability to solve a 8 bit even parity problem and were provided with 8 inputs and 1 output. As in related studies (e.g. [18]), the function list of the nodes included the following four logic operators \([\text{AND}, \text{NAND}, \text{OR}, \text{and NOR}]\). Programs were evaluated on \(2^8\) patterns, i.e. on all possible input patterns. The \(\lambda\) and the mutrate parameters was set to 4 and to 2\%, respectively. The evolutionary process was terminated when a candidate solution achieved optimal performance or after a total of 1 million candidate solutions were evaluated. The parameter \(N\) was varied systematically in the range [100–1000] within multiple experiments. The fitness of a candidate solution was computed on the basis of the inverse of the offset between the outputs produced by the program and the desired outputs. More specifically, the fitness is calculated on the basis of the following equation:

\[
F = 1 - \frac{1}{2^n} \sum_{j=1}^{2^n} |O_j - E_j|
\]

where \(n\) is the number of inputs of the circuit, \(j\) is the number of the input patterns varying in the range \([1,2^n]\), \(O_j\) is the output of the circuit for pattern \(j\), \(E_j\) is the desired output for pattern \(j\).

In the case of the regression problem, we used the Pagie function (Fig. 2) that constitutes a challenging benchmark [22, 29]. Following Turner and Miller [29], programs are provided with two inputs, 1 output and \(N\) nodes. The parameter \(N\) was varied systematically in the range [50–200] in multiple experiments. The input patterns consist of 676 samples taken randomly from the range \(x_1\) in \([-5.0, 5.0]\) and \(x_2\) in \([-5.0, 5.0]\) with a uniform random distribution, and the function set contains the following operators: \([+, -, *, \text{and} /]\). The fitness encodes the regression loss, i.e. the sum of the absolute errors produced by the program. To measure the fraction of times in which the evolutionary process manages to solve the problem we consider the problem solved when the total error is lower than 0.1. The continuous nature of this problem implies that the probability that offspring have identical fitness is much smaller in this problem than in the parity problem.

\[
f(x_1, x_2) = \frac{1}{1 + x_1^4} + \frac{1}{1 + x_2^4}.\]

To automatically adapt the mutation rate to the characteristics of the problem and to the specific evolutionary phase
we used the one-fifth success rule that automatically tunes the mutation rate so to maintain the ratio of offspring that equal or outperform the parent to 1/5 [24–26]. This is realized by varying the mutation rate after the evaluation of each offspring on the basis of the following equation:

\[
M_{\text{rate}} = \begin{cases} 
M_{\text{rate}} \times 1.4 & \text{if } f(\text{offsp.}) \geq f(\text{parent}) \\
M_{\text{rate}} \times 1.4^{-\frac{1}{4}} & \text{otherwise}
\end{cases}
\] (3)

We will refer to the variants of the two algorithms described above that include the automatic adaptation of the mutation rate with the name (1 + 4) ES-AM and (1 + 4) ES-PL-AM.

The ratio of offspring that equal or outperform their parent depends on the mutation rate, i.e. the higher the mutation rate is the lower the probability that offspring equal or outperform their parent is. Moreover it depends on the following factors: (i) the local fitness surface (the higher the derivative of the local fitness surface is, the higher the chance that mutations cause maladaptive effects is), (ii) the number of functional genes (the larger the number of functional genes is, the higher the probability that mutations have maladaptive effect is), and (iii) the robustness to mutations (the higher the robustness of the current parent to mutations is, the lower the probability that mutations cause a fitness loss is). The one-fifth success rule permits to appropriately tune the mutation rate to the variations of these factors across generations.

The source code for replicating the experiments can be downloaded from: https://laral.istc.cnr.it/res/preferentiallylarge.

5 Results

5.1 The parity problem

Table 1 reports the results of 20 series of experiments performed with the (1 + 4) ES in which we systematically varied the size of the genotype (i.e. the number of nodes) and the mutation rate. As can be seen, the value of these two hyperparameters have a strong impact on performance.

This can be explained by considering that that length of the genotype influences the size of the functional circuit of evolving candidate solutions (Table 1, data indicated within parenthesis) that in turns influences the evolvability of candidate solutions. Moreover, it can be explained by considering that the optimal mutation probability is influenced by the size of the functional circuit. This since, the higher the size of the functional circuit is, the higher the probability that random variations alter the functional circuit is. The problem is further complicated by the fact that the size of the functional circuit can change significantly over generations (see Fig. 3).

The (1 + 4) ES-AM method represents a good solution to these problems since it releases the experimenter from the burden of setting the mutation rate hyperparameter, permits to adapt the mutation rate dynamically during the course of the evolutionary process, and permits to achieve performance that are better or at least equally good with respect to the performance obtained by the (1 + 4) ES method with the mutation rate optimized for each condition (Table 2). Indeed, the (1 + 4) ES-AM method permits to achieve high performance without manually optimizing the mutation rate providing that the genotype includes at least 400 gates. The performance of the ES-AM method does not statistically differ from the performance of the ES method with optimal mutation rate in all cases (Mann–Whitney U-test p value > 0.05).

The fact that the (1 + 4) ES-AM-PL method outperforms the (1 + 4) ES-AM when the length of the genotype is 100 or 200 (Mann–Whitney U-test p value < 0.001) and produce equally good performance in the other cases (Mann–Whitney U-test p value > 0.05) demonstrates that the preferential selection of larger phenotype is advantageous when the size of the genotype is sub-optimal.

5.2 The Paige regression problem

In this section we report the results obtained on the Paige regression problem. In consideration of the efficacy of the adaptive mutation method illustrated above we included this technique in all experiments reported below.

The preferential selection of larger solutions permits to achieve better performance also in the case of the Paige regression problem (Table 3). Indeed, the performance achieved by the (1 + 4) ES-AM-PL method are significantly better than the performance achieved by the (1 + 4) ES-AM method (Mann–Whitney U-test p value < 0.001) in all cases.

| Table 1 Performance of the best circuits evolved with the (1 + 4) ES obtained by varying the number of nodes and the mutation rate |
| MuRate 1% | MuRate 2% | MuRate 3% | MuRate 4% |
|-----------|-----------|-----------|-----------|
| Nodes 100 | 0% (32.1) | 35% (35.3) | 10% (33.7) | 5% (31.6) |
| Nodes 200 | 55% (42.4) | 35% (41.6) | 20% (38.7) | 10% (36.9) |
| Nodes 400 | 70% (49.3) | 50% (47.9) | 25% (46.8) | 10% (45.5) |
| Nodes 600 | 70% (56.3) | 40% (55.9) | 10% (54.2) | 0% (53.6) |
| Nodes 1000| 70% (65.4) | 20% (66.8) | 5% (64.3)  | 0% (62.3) |

The number in each cell indicates the percentage of replications that found an optimal solution over 30 replications after one million of evaluations which correspond to 200.000 generations. The number in parentheses indicate the average size of the functional circuits at the end of the evolutionary process.
The fraction of replications that successfully solve the problem, however, is rather small. To verify whether a less greedy selection strategy could improve performance we designed a quasi-neutral (QN) variation of the algorithm that replace the parent with the best offspring even when the performance of the offspring is lower than the parent within a threshold of 10%. We refer to this method as quasi-neutral since it retains also offspring that received quasi-neutral counter-adaptive mutations. A similar effect could be obtained by using stochastic selection methods such as roulette-wheels, tournament selection or stochastic universal sampling. Table 3 reports the results obtained with two variations of this algorithm: (1 + 4) ES-AM-QN and (1 + 4) ES-AM-QN-PL. Both variations include adaptive mutations. The latter also include the preferential selection of large solution among solutions that vary in performance for less than 10%.

As shown in Table 3, the utilization of a quasi-neutral selection scheme and the utilization of the preferential selection of larger solutions lead to significantly better results. Indeed, the performance obtained with the (1 + 4) ES-AM-QN method are significantly better than the performance obtained with the (1 + 4) ES-AM-PL and (1 + 4) ES-AM methods (Mann–Whitney U-test p value < 0.001). Moreover, the performance obtained with the (1 + 4) ES-AM-QN-PL method are significantly better than the performance obtained with all other methods (Mann–Whitney U-test p value < 0.001).

The utilization of quasi-neutral selection scheme did not provide an advantage in the case of the parity problems (results not shown). This can be explained by considering that the probability to produce neutral offspring, i.e. offspring achieving the same fitness of their parent, is much higher in the parity problem than in the Paige regression problem.

6 Conclusions

In this paper we demonstrated how the efficiency of Cartesian genetic programming methods can be enhanced through the usage of adaptive mutation and through the preferential selection of phenotypically larger solutions.

The rationale behind the method proposed is that evolution tends to select solutions that are robust with respect to mutations and that the simplest way to achieve robustness consists in selecting programs that use a minimal number of nodes. This since the smaller the number of functional nodes is, the smaller the probability that these nodes are altered by mutations is. As pointed out in previous research [14, 23] the chance to produce phenotypically different programs as a result of genetic variations is positively correlated with the size of the solutions. Consequently, the tendency to select minimal solutions that are very robust to mutations reduces the probability to generate better solutions as a result of mutations which, in turn, reduces the efficacy of
The results with the former method refer to those obtained by setting the mutation rate to the value that resulted optimal in each condition (indicated in square brackets). The first raw reports the same data described in Table 1 for experiments with 100–1000 Nodes. In the case of the experiment carried with the other methods the initial mutation rate was set to 2%. Each data indicates the average results of 30 replications.

The number in each cell indicates the percentage of replications that resulted optimal in each condition (indicated in square brackets). The first raw reports the same data described in Table 1 for experiments with 100–1000 Nodes. In the case of the experiment carried with the other methods the initial mutation rate was set to 2%. Each data indicates the average results of 30 replications.

Table 2: Performance obtained with the (1 + λ) ES and with the (1 + λ)-ES-AM algorithms in experiments carried with genotype of different length

| CGP Nodes | 50     | 100    | 200     |
|-----------|--------|--------|---------|
| (1 + 4) ES | 0% [1%] | 35% [2%] | 55% [1%] |
| (1 + 4) ES-AM | 0%    | 30%    | 30%     |
| (1 + 4) ES-AM-PL | 0%    | 75%    | 80%     |

The number in each cell indicates the percentage of replications that resulted optimal in each condition (indicated in square brackets). The first raw reports the same data described in Table 1 for experiments with 100–1000 Nodes. In the case of the experiment carried with the other methods the initial mutation rate was set to 2%. Each data indicates the average results of 30 replications.

The advantage provided by the preferential selection of larger solutions can be further extended by self-adapting the mutation rate through the one-fifth success rule [24–26]. The advantage of the combined effect of the preferential selection of larger solutions and of self-adaptation of the mutation rate is due to the fact that the robustness of evolving solutions is strongly influenced by the size of their functional circuit. Consequently, the dynamic adaptation of the mutation rate becomes especially important when the functional size of the evolving solutions varies significantly among generations.

Recent works have demonstrated the applicability of Cartesian genetic programming also to problems that are more complex with respect to those used in this paper (see Miller [21] for a review). We restricted our analysis to two relatively complex problems to permit an easier comparison with results obtained with standard methods and to evaluate the efficacy of the method proposed on two qualitatively different domains. Future works might investigate the applicability of our methods to more complex problems and the potential advantages of more elaborated mutation methods such as the single active mutation [5].

Acknowledgements: The authors thanks Prof. Julian Miller for insightful discussions and suggestions.

References

1. Edelman GM, Gally JA (2001) Degeneracy and complexity in biological systems. Proc Natl Acad Sci USA 98(13):763–768
2. Garmendia-Doval AB, Morley DS, Juhos S (2003) Post docking filtering using cartesian genetic programming. In:Liartdet P (ed) Proceedings of the 6th international conference on artificial evolution, pp 435–446
3. Gajda Z, Sekanina L (2007) Reducing the number of transistors in digital circuits using gate-level evolutionary design. In: Genetic and evolutionary computation conference, ACM, pp 245–252
4. Gajda Z, Sekanina L (2007) An efficient selection strategy for digital circuit evolution. In: International conference on evolvable systems: from biology to hardware. Lecture Notes on Computer Science 6274, pp 13–24
5. Goldman B.W., and Punch W.F. (2013). Reducing wasted evaluations in cartesian genetic programming. In: Proceedings of the 12th European conference on genetic programming. Springer, Berlin
6. Harding S, Miller JF (2005) Evolution of robot controller using cartesian genetic programming. In: Proceedings of the 8th European conference on genetic programming, vol 3447. Lausanne, Switzerland, pp 62–73
7. Harding S, Graziano V, Leitner J, Schmidhuber J (2012) MT-CGP: mixed type Cartesian genetic programming. In: Proceedings of the 14th genetic and evolutionary computation conference (GECCO). Philadelphia, PA, USA, pp 751–758
8. Harding S, Miller J, Banzhaf W (2009) Self-modifying Cartesian genetic programming: fibonacci, squares, regression and summing. In: Proceedings of the 12th European conference on
genetic programming (EuroGP), vol 5481.Tuebingen, Germany, pp 133–144
9. Hu T, Payne JL, Banzhof W, Moore JH (2012) Evolutionary dynamics on multiple scales: a quantitative analysis of the interplay between genotype, phenotype, and fitness in linear genetic programming. Genet Program Evolvable Mach 13(3):305–337
10. Khan MM, Ahmad AM, Khan GM, Miller JF (2013) Fast learning neural networks using Cartesian genetic programming. Neurocomputation 121:274–289
11. Kalganova T, Miller JF (1999) Evolving more efficient digital circuits by allowing circuit evolution on multi-objective fitness. In: The first NASA/DoD workshop on evolvable hardware, IEEE Computer Society, pp 54–63
12. Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge
13. Langdon WB, Poli R (2002) Foundations of genetic programming. Springer, Berlin
14. Milano N, Pagliuca P, Nolfi S (2019) Robustness, evolvability and phenotypic complexity: insights from evolving digital circuits. Evol Intell 12(1):83–95
15. Milano N, Nolfi S (2016) Robustness to faults promotes evolvability: insights from evolving digital circuits. PLoS ONE 11(7):e0158627
16. Miller JF (2011) Cartesian genetic programming. Springer, Berlin
17. Miller JF, Smith SL (2006) Redundancy and computational efficiency in Cartesian genetic programming. IEEE Trans Evolut Comput 10(2):167–174
18. Miller JF, Thomson P (2000) Cartesian genetic programming. In: Proceedings of the third European conference on genetic programming (EuroGP), vol 1820. Springer, Berlin, pp 121–132
19. Miller JF, Job D, Vassilev VK (2000) Principles in the evolutionary design of digital circuits—Part I. J Genet Program Evolvable Mach 1:8–35
20. Miller JF, Job D, Vassilev VK (2000) Principles in the evolutionary design of digital circuits—Part II. J. Genet Program Evolvable Mach 1:259–288
21. Miller JF (2019) The alchemy of computation: designing with the unknown. Nat Comput 1–12
22. Pagie L, Hogeweg P (1997) Evolutionary consequences of coevolving targets. Evolut Comput 5(4):401–418
23. Raman K, Wagner A (2011) The evolvability of programmable hardware. J R Soc Interface 8(55):269–281
24. Rechenberg I (1973) Evolutionsstrategie. Friedrich Frommann Verlag (Gunther Holzboog KG), Stuttgart
25. Rechenberg I (1973) Evolutionstrategie—Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog, Stuttgart
26. Schumer M, Steiglitz K (1968) Adaptive step size random search. IEEE Trans Autom Control 13:270–276
27. Schuster P, Fontana W, Stadler PF, Hofacker IL (1994) From sequences to shapes and back: a case study in RNA secondary structures. Proc R Soc Lond B 255:279–284
28. Tononi G, Sporns O, Edelman GM (1999) Measures of degeneracy and redundancy in biological networks. Proc Nat Acad Sci USA 96:3257–3262
29. Turner JT, Miller JF (2015) Neutral genetic drift: an investigation using Cartesian genetic programming. Genet Program Evolvable Mach 16(4):531–558
30. Turner AJ, Miller JF (2014) Recurrent cartesian genetic programming. In: 13th International conference on parallel problem solving from nature (PPSN 2014), LNCS, vol 8672. Springer, Berlin, pp 476–486
31. Van Nimwegen E, Crutchfield JP, Huynen M (1999) Neutral evolution of mutational robustness. PNAS 96:9716–9720
32. Vassilev V, Job D, Miller J (2000) Towards the automatic design of more efficient digital circuits. In: Proceedings of the 2nd NASA/DoD workshop on evolvable hardware, IEEE Computer Society, pp 151–160
33. Wagner A (2008) Robustness and evolvability: a paradox resolved. Proc R Soc B 275:91–100
34. Wagner A (2011) The origins of evolutionary innovations: a theory of transformative change in living systems. Oxford University Press, Oxford
35. Wilson DG, Cussat-Blanc S, Luga H, Miller JF (2018) Evolving simple programs for playing Atari games. arXiv preprint arXiv:1806.05695

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.