General Boolean function benchmark suite

Kalkreuth, R.T.; Vasicek, Z.; Husa, J.; Vermetten, D.L.; Ye, F.; Bäck, T.H.W.

Towards a General Boolean Function Benchmark Suite

Roman Kalkreuth
r.t.kalkreuth@liacs.leidenuniv.nl
Leiden Institute of Advanced Computer Science, Leiden University
Leiden, Netherlands

Zdeněk Vašíček
vasicek@fit.vut.cz
Brno University of Technology
Brno, Czech Republic

Jakub Husa
ihusa@fit.vut.cz
Brno University of Technology
Brno, Czech Republic

Diederick Vermetten
d.l.vermetten@liacs.leidenuniv.nl
Leiden Institute of Advanced Computer Science, Leiden University
Leiden, Netherlands

Furong Ye
f.ye@liacs.leidenuniv.nl
Leiden Institute of Advanced Computer Science, Leiden University
Leiden, Netherlands

Thomas Bäck
T.H.W.Baeck@liacs.leidenuniv.nl
Leiden Institute of Advanced Computer Science, Leiden University
Leiden, Netherlands

ABSTRACT

Just over a decade ago, the first comprehensive review on the state of benchmarking in Genetic Programming (GP) analyzed the mismatch between the problems that are used to test the performance of GP systems and real-world problems. Since then, several benchmark suites in major GP problem domains have been proposed over time, which were able to fill some of the major gaps. In the framework of the first review about the state of benchmarking in GP, logic synthesis was classified as one of the major GP problem domains. However, a diverse and accessible benchmark suite for logic synthesis is still missing in the field of GP. In this work, we take a first step towards a benchmark suite for logic synthesis that covers different types of Boolean functions that are commonly used for the evaluation of GP systems. We also present baseline results that have been obtained by former work and in our evaluation experiments by using Cartesian Genetic Programming.

CCS CONCEPTS

• **Computing methodologies** → **Discrete space search**; **Genetic programming**; **Evolvable hardware**; • **Hardware** → **Combinational synthesis**.

KEYWORDS

Benchmarking, Boolean function learning, Genetic Programming

ACM Reference Format:
Roman Kalkreuth, Zdeněk Vašíček, Jakub Husa, Diederick Vermetten, Furong Ye, and Thomas Bäck. 2023. Towards a General Boolean Function Benchmark Suite. In *Genetic and Evolutionary Computation Conference Companion (GECCO '23 Companion), July 15–19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3583133.3590685

ACKNOWLEDGMENTS

This work was supported by the Czech Science Foundation project 22-02067S.

GECCO '23, July 15–19, 2023, Lisbon, Portugal

ACM ISBN 979-8-4007-0120-7/23/07.
https://doi.org/10.1145/3583133.3590685

1 INTRODUCTION

A little over a decade ago, McDermott et al. [20] published the first review on benchmarking in GP [3, 4, 6, 13], which highlighted the damaging gap between benchmarks commonly used to test GP systems and real-world problems. The effort of McDermott et al. did not go beyond a following community survey [32] and ultimately proposed a new benchmark suite for GP but various benchmarks in major GP problem domains such as program synthesis, symbolic regression or classification emerged afterward [5, 23, 24]. At last year's GECCO conference, the work of McDermott et al. [20] was awarded the SIGEVO Impact award, which triggered reflection on the developments of the last decades. Very recently, a follow-up article on the state and of development of Benchmarking has been published by McDermott et al. [19]. Besides reviewing well-established GP benchmark suites which have been proposed over the last years, the missing of a Boolean function benchmark suite for logic synthesis (LS) has been identified as one of the major gaps. Although various benchmarks for LS have been proposed in the past, a general benchmark suite for LS is still missing. Moreover, the authors state that for the future of benchmarking in LS, it might be useful to further increase the diversity of benchmarks by exploring new Boolean function problems and curating these problems into a new benchmark suite.
Therefore, in this work, we follow up the suggestion of McDermott et al. [19] and consider benchmarking in LS from a general perspective. We reflect on the requirements for a general benchmark suite for LS, and bundle together a set of Boolean functions from the major categories commonly used in previous work on GP. The selected benchmarks aim at the synthesis of Boolean functions from scratch, which has been considered a challenging task for GP systems [30].

2 LOGIC SYNTHESIS IN GENETIC PROGRAMMING

Logic synthesis (LS) in GP can be considered a black-box and optimization problem domain that has played a major role in the application scope of GP research throughout its history. In general, LS by means of GP refers to the application of GP models to synthesize expression that match the input-output mapping of Boolean functions. Boolean functions can be formally expressed

and mathematically described with Boolean expressions. LS as tackled with GP paradigm predominantly addresses two major tasks in this problem domain:

(1) Synthesis of a Boolean expression that produces the correct output given the inputs of the Boolean function.
(2) Optimization of the Boolean expression that represents a certain Boolean function.

The latter task is approached by defining one or more optimization objectives. Both tasks are performed in accordance with Boolean logic and algebra. Besides, algebraic expressions, Boolean functions are commonly represented with truth tables that describe the input-output mapping of the respective function.

2.1 Learning of Boolean Functions with Known Input-output Mapping

As a nature-inspired search heuristic for automatic programming, GP is well suited for LS, since various GP representation models can easily be applied to this task. LS as an application field for GP was popularized by Koza by representing LISP programs as parse trees [13–17]. Since various Boolean expressions can be formulated as LISP S-expressions, Koza used his approach to evolve Boolean expressions for Boolean functions like digital multiplexers and parity.

Since Koza's parse-tree representation tree-based model of GP aims for single-output functions, further work in GP concentrated on graph-based multiple-output representation models [10, 21, 22, 27]. Commonly used multiple-output functions for the evaluation and comparison of graph-based GP models and corresponding operators are arithmetic functions such as the digital adder and multiplier as well as combinational functions [1, 8, 12, 29]. Since a large part of Boolean functions can be implemented as digital circuits a real-world application of graph-based GP is located in the field of evolvable hardware [28]. This type of application poses the requirement on GP systems to be able to synthesize combinational circuits from scratch which has been considered to be a challenging task for graph-based GP [30].

2.2 Learning of Cryptographic Boolean Functions

The majority of problems in LS have been specified by the known input-output mapping, such as the truth table. However, there are problems where the input-output mapping is not known a priori. These are known as black box problems, because the target Boolean function can only be interacted with through its inputs and outputs. Cryptographic Boolean functions fall into this category. They are designed and analyzed on the basis of their specific properties such as nonlinearity, algebraic degree or correlation immunity, each of which makes them resistant against certain types of cryptographic attacks [2]. The goal of the heuristic search is then to find any function that possesses a certain set of these properties.

Applications of EC have focused mainly on Boolean functions with high nonlinearity and some combinations of additional properties, suitable for use in stream ciphers [9, 25, 26], and functions with low Hamming weight suitable for protecting cryptographic applications against side-channel attacks on their implementation [25].

3 TOWARDS A BENCHMARK SUITE FOR BOOLEAN FUNCTION SYNTHESIS

3.1 General Motivation

The primary motivation for our work is to promote the synthesis of Boolean functions in GP research, since a general benchmark suite for this task is still missing. Another general motivation behind our benchmark suite is to promote diversity and accessibility in the Boolean problem domain, which we also identified as a gap in the field. We also think that a benchmark suite on logic synthesis should include the scaling property of Boolean functions, since it has been commonly used to evaluate the robustness of GP methods in the past. For instance, the upscaling of the bit-length can be used to increase the complexity of a Boolean function, but similarity is maintained.

3.2 Main Objectives and Properties

Considering our general motivation, we think that the philosophy behind a LS benchmark suite should respect the following objectives and properties: **Generalization**, **Accessibility**, **Scaling** and **Blacklisting**. A generalized approach to benchmarking in LS is achieved by covering a broader spectrum of types of Boolean functions. However, our approach is balanced with practical orientation since all proposed functions can be implemented in digital circuits. Another property of the benchmark suite is to enable accessibility to the data of the respective benchmarks by providing interfaces for the used formats in three major programming languages: C++, Java and Python. We make use of the scaling property of Boolean functions by proposing benchmarks of the same type with different bit lengths of the respective inputs. Scaling up the bit length increases the difficulty of the proposed benchmarks but maintains similarity. It has been found that the overuse of single-output functions affects diversity in LS negatively [20, 32]. Moreover, since the overused low-order parity-even and multiplexer benchmarks have been blacklisted, we only include high-order parity functions and promote diversity by emphasizing multiple-output Boolean functions as it was recommended by White et al. [32].

3.3 Problem Selection

Based on a comprehensive survey of relevant work published in the last two decades, we propose to include the following problems in the benchmark suite:

3.3.1 Adders. Adders are a popular arithmetic benchmark that has been used within the EC community since the early days. Adders are naturally targeted as a more viable alternative to the evolution of multipliers.

3.3.2 Multipliers. In addition to adders, we also include multipliers. The evolutionary design of multipliers represents probably the hardest problem due to the complexity of the multiplication itself (the multipliers consist of a sequence of adders reducing the partial products to a single output vector) [7].

3.3.3 Demultiplexer. The demultiplexer benchmark is a multiple-output problem which has been used in former work to evaluate the search performance of graph-based GP systems [1, 11, 12, 31].

Table 1: Function sets used for the evaluations

Identifier	Functions	Description
$\mathcal{F}_{\ominus}$	AND, OR, NAND, NOR	Reduced function set
$\mathcal{F}_{\oplus}$	BUFa, NOTa, AND, OR, XOR, NAND, NOR, XNOR	Extended function set
$\mathcal{F}_{p}$	BUFa, NOTa, AND, OR	Parity
$\mathcal{F}_{c}$	AND, XOR, OR, XNOR, INHb	Cryptographic

3.3.4 Comparators. We include two types of comparators, the identity and magnitude comparator. Both types check whether one value is less or greater than the other or if both values are equal. The identity comparator is another benchmark which provides a scalable multiple-output problem that has been proposed by Walker and Miller [31]. The identity comparator makes binary comparisons among the inputs, and each input represents an operand. The magnitude comparator compares the sum of two operands, whereby the size of each operand depends on the respective bit length.

3.3.5 Mixed (building blocks). Mixed or multi-functional Boolean functions have been comparatively less used in the past. In the field of CGP, experiments have predominantly concentrated on the evaluation of single-functional benchmarks. However, Walker and Miller [31] proposed a 3-bit arithmetic logic unit (ALU) benchmark for the evaluation of a multi-chromosome approach to the CGP representation model. A detailed specification of this type of benchmark is available in the GitHub repository.

3.3.6 Parity circuits. Parity circuits are another commonly used benchmark. Although the construction of an optimal parity circuit is a straightforward process, these circuits provide a simple, yet challenging problem that can be used to evaluate and compare different optimization algorithms. The difficulty comes from the fact that Parity is a symmetric Boolean function where the output only depends on the number of ones. 5,8,10 and 12-input parities have been used to investigate the role of neutrality in CGP [33].

3.3.7 Cryptographic functions. To support the diversity of the benchmark set, we also included four types of cryptographic Boolean functions – bent, balanced, resilient, and masking – each with different levels of complexity and relevant properties [2, 25].

3.3.8 Function sets. Since the majority of the problems are typically implemented with XOR gates, we propose the use of a reduced function set $\mathcal{F}_{\ominus}$ (see Table 1) for the easier problems to increase the difficulty of these problems. For the evaluation of more complex problems, we recommend an extended function set $\mathcal{F}_{\oplus}$. For the evaluation of the cryptographic benchmarks the set $\mathcal{F}_{c}$ should be used [25, 26].

3.4 Evaluation Methods (Fitness Calculation)

3.4.1 Similarity of Boolean Circuits. In the case of Boolean circuit evolution, the Hamming distance is typically used in the fitness function to determine the similarity of a Boolean circuit encoded by a candidate solution to the specification. Typically, the problem specification is provided in the form of a truth table, which defines the output of the circuit for every possible input combination. The Hamming distance then measures the difference between the desired and actual outputs of a circuit. As the distance corresponds to the number of positions in which two Boolean vectors differ, its usage in the fitness function is straightforward, as it naturally assigns a lower score to circuits that produce outputs closer to the desired outputs and a higher score to circuits that produce outputs further away from the desired outputs. The circuit with the lowest Hamming distance is considered the best and selected for the next generation.

3.4.2 Evaluation of cryptographic properties. Cryptographic benchmarks consider five properties, which are defined as follows. Hamming weight of a Boolean function is defined as the number of ones in its truth table [2]. A function is said to be balanced if its truth table contains the same amount of ones and zeros. That is, if its Hamming weight is equal to $HW(f) = 2^{n-1}$ [2].
To evaluate nonlinearity, the truth table of a function needs to be converted into a Walsh spectrum $WS(f)$, which defines nonlinearity as $NL(f) = 2^{n-1} - \frac{1}{2}max(WS(f))$ and represents the Hamming distance between the function and the nearest affine (linear or inversion of a linear) function [2].
Correlation immunity is also defined using Walsh spectrum. If values all items of the Walsh spectrum with Hamming weight $1 \leq HW \leq t$ are zero, then the Boolean function has correlation immunity of degree t [2].
To evaluate algebraic degree, the truth table of a function needs to be converted into algebraic normal form (ANF). Boolean function has algebraic degree of d if its ANF contains at least one item with Hamming weight $HW \geq d$ [2].
For the purposes of heuristic search, all properties are normalized to range <0, 1>, those which are relevant to the specific type of a cryptographic function are added together, and the candidate solution with the highest score is considered the best.

3.5 Interfaces and Resources

The data files for the benchmarks are available in several formats and publicly accessible in our GitHub repository[1]. For each benchmark, we provide open-source parameterized Verilog model allowing to easily scale the bit width, synthesized baseline in common BLIF format as well as the complete input-output mapping in uncompressed and compressed form for ease of use. Due to the limited space, the data formats are described in more details in the repository. Interfaces for C++, Java and Python can also be found in our repository.

3.6 Benchmarks and Baseline Results

The baselines results for the selected benchmarks have been obtained with standard CGP in former work and in our experiments. We used the common $1 + \lambda$ algorithm with standard probabilistic point mutation and neutral genetic drift. To report fair results with CGP, we performed hyperparameter optimization (HPO) with irace [18] for each tested benchmark. For the evaluation of each benchmark, we performed 100 runs and measured the number of fitness evaluations until the CGP algorithm terminated. The list of benchmarks and the corresponding baseline results are provided in our GitHub repository.

[1] https://github.com/boolean-function-benchmarks

4 CONCLUSIONS AND FUTURE WORK

In this work, we made the first step towards an accessible and diverse benchmark suite for logic synthesis that will consist of 32 benchmarks selected from popular categories of Boolean functions. Overall, the selected benchmarks cover a wide range of Boolean functions in terms of problem hardness and input-output ratio. The use of the selected benchmarks can support the evaluation of the search performance and robustness of GP methods in LS. Naturally, our next step will be to propose a new benchmark suite for LS. Another point which will be addressed by future work concerns the study of the search complexity of our selected benchmarks. More precisely, we plan to propose complexity measurements that are based on the data of the benchmarks and to study the correlation to the input-output ratio of the benchmarks.

REFERENCES

[1] Timothy Atkinson, Detlef Plump, and Susan Stepney. 2018. Evolving Graphs by Graph Programming. In *Genetic Programming - 21st European Conference, EuroGP 2018, Parma, Italy, April 4-6, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10781)*, Mauro Castelli, Lukás Sekanina, Mengjie Zhang, Stefano Cagnoni, and Pablo García-Sánchez (Eds.). Springer, 35–51. https://doi.org/10.1007/978-3-319-77553-1_3

[2] Ann Braeken. 2006. *Cryptographic properties of Boolean functions and S-boxes*. Ph. D. Dissertation. Katholieke Universiteit Leuven.

[3] Nichael Lynn Cramer. 1985. A Representation for the Adaptive Generation of Simple Sequential Programs. In *Proceedings of the 1st International Conference on Genetic Algorithms, Pittsburgh, PA, USA, July 1985*, John J. Grefenstette (Ed.). Lawrence Erlbaum Associates, 183–187.

[4] Richard Forsyth. 1981. BEAGLE A Darwinian Approach to Pattern Recognition. *Kybernetes* 10, 3 (1981), 159–166. https://doi.org/doi:10.1108/eb005587

[5] Thomas Helmuth and Lee Spector. 2015. General Program Synthesis Benchmark Suite. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015*, Sara Silva and Anna Isabel Esparcia-Alcázar (Eds.). ACM, 1039–1046. https://doi.org/10.1145/2739480.2754769

[6] Joseph Hicklin. 1986. *Application of the Genetic Algorithm to Automatic Program Generation*. Master's thesis. University of Idaho.

[7] David Hodan, Vojtech Mrazek, and Zdenek Vasicek. 2020. Semantically-Oriented Mutation Operator in Cartesian Genetic Programming for Evolutionary Circuit Design. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference* (Cancún, Mexico) *(GECCO '20)*. Association for Computing Machinery, New York, NY, USA, 940–948. https://doi.org/10.1145/3377930.3390188

[8] David Hodan, Vojtech Mrazek, and Zdenek Vasícek. 2021. Semantically-oriented mutation operator in cartesian genetic programming for evolutionary circuit design. *Genet. Program. Evolvable Mach.* 22, 4 (2021), 539–572. https://doi.org/10.1007/s10710-021-09416-6

[9] Jakub Husa. 2019. Comparison of genetic programming methods on design of cryptographic boolean functions. In *European Conference on Genetic Programming*. Springer, 228–244.

[10] T. Kalganova. 1997. Evolutionary Approach to Design Multiple-valued Combinational Circuits. In *Proceedings. of the 4th International conference on Applications of Computer Systems (ACS'97)*. Szczecin, Poland, 333–339.

[11] Roman Kalkreuth. 2019. Two New Mutation Techniques for Cartesian Genetic Programming. In *Proceedings of the 11th International Joint Conference on Computational Intelligence, IJCCI 2019, Vienna, Austria, September 17-19, 2019*, Juan Julián Merelo Guervós, Jonathan M. Garibaldi, Alejandro Linares-Barranco, Kurosh Madani, and Kevin Warwick (Eds.). ScitePress, 82–92. https://doi.org/10.5220/0008070100820092

[12] Roman Kalkreuth. 2022. Towards Phenotypic Duplication and Inversion in Cartesian Genetic Programming. In *Proceedings of the 14th International Joint Conference on Computational Intelligence, IJCCI 2022, Valletta, Malta, October 24-26, 2022*, Thomas Bäck, Bas van Stein, Christian Wagner, Jonathan M. Garibaldi, H. K. Lam, Marie Cottrell, Faiyaz Doctor, Joaquim Filipe, Kevin Warwick, and Janusz Kacprzyk (Eds.). SCITEPRESS, 50–61. https://doi.org/10.5220/0011551000003332

[13] John R. Koza. 1989. Hierarchical Genetic Algorithms Operating on Populations of Computer Programs. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989*, N. S. Sridharan (Ed.). Morgan Kaufmann, 768–774. http://ijcai.org/Proceedings/89-1/Papers/123.pdf

[14] John R. Koza. 1990. Concept Formation and Decision Tree Induction Using the Genetic Programming Paradigm. In *Parallel Problem Solving from Nature, 1st Workshop, PPSN I, Dortmund, Germany, October 1-3, 1990, Proceedings (Lecture Notes in Computer Science, Vol. 496)*, Hans-Paul Schwefel and Reinhard Männer (Eds.). Springer, 124–128. https://doi.org/10.1007/BFb0029742

[15] John R. Koza. 1990. *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems*. Technical Report. Stanford, CA, USA.

[16] John R. Koza. 1990. A Hierarchical Approach to Learning the Boolean Multiplexer Function. In *Proceedings of the First Workshop on Foundations of Genetic Algorithms. Bloomington Campus, Indiana, USA, July 15-18 1990*, Gregory J. E. Rawlins (Ed.). Morgan Kaufmann, 171–192. https://doi.org/10.1016/b978-0-08-050684-5.50014-8

[17] John R. Koza. 1993. *Genetic programming - on the programming of computers by means of natural selection*. MIT Press.

[18] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. 2016. The irace package: Iterated Racing for Automatic Algorithm Configuration. *Operations Research Perspectives* 3 (2016), 43–58. https://doi.org/10.1016/j.orp.2016.09.002

[19] James McDermott, Gabriel Kronberger, Patryk Orzechowski, Leonardo Vanneschi, Luca Manzoni, Roman Kalkreuth, and Mauro Castelli. 2022. Genetic Programming Benchmarks: Looking Back and Looking Forward. *SIGEVOlution* 15, 3, Article 1 (dec 2022), 19 pages. https://doi.org/10.1145/3578482.3578483

[20] James McDermott, David Robert White, Sean Luke, Luca Manzoni, Mauro Castelli, Leonardo Vanneschi, Wojciech Jaskowski, Krzysztof Krawiec, Robin Harper, Kenneth A. De Jong, and Una-May O'Reilly. 2012. Genetic programming needs better benchmarks. In *Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11, 2012*, Terence Soule and Jason H. Moore (Eds.). ACM, 791–798. https://doi.org/10.1145/2330163.2330273

[21] Julian F. Miller. 1999. An empirical study of the efficiency of learning boolean functions using a Cartesian Genetic Programming approach. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith (Eds.), Vol. 2. Morgan Kaufmann, Orlando, Florida, USA, 1135–1142. http://citeseer.ist.psu.edu/153431.html

[22] J. F. Miller, P. Thomson, and T. Fogarty. 1997. Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study. In *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*. Wiley, 105–131.

[23] Patryk Orzechowski, William La Cava, and Jason H. Moore. 2018. Where Are We Now? A Large Benchmark Study of Recent Symbolic Regression Methods. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Kyoto, Japan) *(GECCO '18)*. Association for Computing Machinery, New York, NY, USA, 1183–1190. https://doi.org/10.1145/3205455.3205539

[24] Patryk Orzechowski and Jason H. Moore. 2021. Generative and reproducible benchmarks for comprehensive evaluation of machine learning classifiers. *CoRR* abs/2107.06475 (2021). arXiv:2107.06475 https://arxiv.org/abs/2107.06475

[25] Stjepan Picek, Claude Carlet, Sylvain Guilley, Julian F Miller, and Domagoj Jakobovic. 2016. Evolutionary algorithms for boolean functions in diverse domains of cryptography. *Evolutionary computation* 24, 4 (2016), 667–694.

[26] Stjepan Picek, Domagoj Jakobovic, Julian F Miller, Lejla Batina, and Marko Cupic. 2016. Cryptographic Boolean functions: One output, many design criteria. *Applied Soft Computing* 40 (2016), 635–653.

[27] Riccardo Poli. 1997. Evolution of Graph-Like Programs with Parallel Distributed Genetic Programming. In *Proceedings of the 7th International Conference on Genetic Algorithms, East Lansing, MI, USA, July 19-23, 1997*, Thomas Bäck (Ed.). Morgan Kaufmann, 346–353.

[28] Lukás Sekanina. 2012. Evolvable Hardware. In *Handbook of Natural Computing*, Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok (Eds.). Springer, 1657–1705. https://doi.org/10.1007/978-3-540-92910-9_50

[29] Léo Françoso Dal Piccol Sotto, Paul Kaufmann, Timothy Atkinson, Roman Kalkreuth, and Márcio Porto Basgalupp. 2021. Graph representations in genetic programming. *Genet. Program. Evolvable Mach.* 22, 4 (2021), 607–636. https://doi.org/10.1007/s10710-021-09413-9

[30] Zdenek Vasicek and Lukas Sekanina. 2014. How to evolve complex combinational circuits from scratch?. In *2014 IEEE International Conference on Evolvable Systems, ICES 2014, Orlando, FL, USA, December 9-12, 2014*. IEEE, 133–140. https://doi.org/10.1109/ICES.2014.7008732

[31] James Alfred Walker, Katharina Völk, Stephen L. Smith, and Julian Francis Miller. 2009. Parallel evolution using multi-chromosome cartesian genetic programming. *Genet. Program. Evolvable Mach.* 10, 4 (2009), 417–445. https://doi.org/10.1007/s10710-009-9093-2

[32] David Robert White, James McDermott, Mauro Castelli, Luca Manzoni, Brian W. Goldman, Gabriel Kronberger, Wojciech Jaskowski, Una-May O'Reilly, and Sean Luke. 2013. Better GP benchmarks: community survey results and proposals. *Genet. Program. Evolvable Mach.* 14, 1 (2013), 3–29. https://doi.org/10.1007/s10710-012-9177-2

[33] Tina Yu and Julian Miller. 2002. Finding Needles in Haystacks Is Not Hard with Neutrality. In *Genetic Programming*, James A. Foster, Evelyne Lutton, Julian Miller, Conor Ryan, and Andrea Tettamanzi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 13–25.