Bridging the Gap Between Evolvable Hardware and Industry Using Cartesian Genetic Programming

Zdenek Vasicek

Abstract Advancements in technology developed in the early nineties have enabled researchers to successfully apply techniques of evolutionary computation in various problem domains. As a consequence, a new research direction referred to as evolvable hardware (EHW) focusing on the use of evolutionary algorithms to create specialized electronics has emerged. One of the goals of the early pioneers of EHW was to evolve complex circuits and overcome the limits of traditional design. Unfortunately, evolvable hardware found itself in a critical stage around 2010 and a very pessimistic future for EHW-based digital circuit synthesis was predicted. The problems solved by the community were of the size and complexity of that achievable in fifteens years ago and seldom compete with traditional designs. The scalability problem has been identified as one of the most difficult problems that researchers are faced with and it was not clear whether there existed a path forward that would allow the field to progress. Despite that, researchers have continued to investigate how to overcome the scalability issues and significant progress has been made in the area of evolutionary synthesis of digital circuits in recent years. The goal of this chapter is to summarize the progress in the evolutionary synthesis of gate-level digital circuits, and to identify the challenges that need to be addressed to enable evolutionary methods to penetrate into industrial practice.

1 Introduction

Advancements in technology developed in the early 1990s have enabled researchers to successfully apply techniques of evolutionary computation in various problem domains. Higuchi et al. [11] and Thompson [30] demonstrated that evolutionary algorithms (EAs) are able to solve non-trivial hardware-related problems. The achievements presented in the seminal paper of Higuchi et al. motivated other scientists to intensively explore a new and promising research topic. As a consequence, a

Z. Vasicek (✉)
Faculty of Information Technology, Centre of Excellence IT4Innovations,
Brno University of Technology, Brno, Czech Republic
e-mail: vasicek@fit.vutbr.cz

S. Stepney and A. Adamatzky (eds.), *Inspired by Nature*, Emergence,
Complexity and Computation 28, https://doi.org/10.1007/978-3-319-67997-6_2

new research direction referred to as *Evolvable hardware* (EHW) has emerged [6]. Evolvable hardware, a field of evolutionary computation, focuses on the use of evolutionary algorithms to create specialized electronics without manual engineering. The vision of EHW is to replace expensive and sometimes unreliable designers and develop robust, flexible and survivable autonomous systems. EHW draws inspiration from three fields: biology, computer science and electronic engineering.

Several schemes have been developed for classifying EHW [6]. Usually, two research areas are distinguished: *evolutionary circuit design* and *evolvable circuits*. In the first case, evolutionary algorithms are used as a tool employed to design a system that meets a predefined specification. For example, genetic programming can be used to discover an area-efficient implementation of a circuit whose function is specified by a truth table. In the second case, an evolutionary algorithm represents an inherent part of an evolvable circuit. The resulting adaptive system is autonomously reconfigured to provide a degree of self-adaptive and self-repair behaviour.

In the context of the circuit design, EHW is an attractive approach as it provides another option to the traditional design methodology: to use evolution to design circuits for us. Moreover, a key strength of the EHW approach is that it can be applied to designing the circuits that cannot be fully specified *a priori*, but where the desired behaviour is known. Another often emphasized advantage of this approach is that circuits can be customized and adapted for a particular environment. For example, if we know that some input combinations in our target application occur with relatively low probability, we can take this information into account and evolve a circuit that shows better parameters such as reduced size, delay or power consumption.

There are several works devoted mainly to the evolvable circuits and adaptive systems [7, 12, 25, 31]. However, there is no similar work systematically mapping the history and progress of evolutionary circuit design. Such a situation may suggest that evolutionary circuit design represents an area in which only marginal results have been achieved. Let us restrict ourselves to the evolution of digital circuits represented at the gate-level, i.e. circuits implemented using common (typically two-input) gates. Considering the complexity of circuits routinely used in industry, we have to admit that only little was done before 2010 even if many novel and promising approaches were published, as surveyed in Sect. 2.1. However, this area has been undergoing a substantial development in the last five years, and many competitive and real-world results have been obtained (Sect. 2.3).

Before we proceed further, it is necessary to clarify what we mean by a complex circuit, as researchers in EC usually have a somewhat illusory notion about circuit complexity. In theoretical computer science, the complexity of Boolean functions is expressed as the *size* or *depth* of corresponding combinational circuits that compute them. The size of a circuit is the number of gates it contains; its depth is the maximal length of a path from an input gate to the output gate. In evolutionary computation, complexity is typically assessed according to the number of input variables, because the number of variables significantly impacts the scalability. In practice, the number of gates is usually considered as a complexity measure. Since the goals of the early pioneers of EHW were to evolve complex circuits and overcome the limits of traditional design, it is necessary to take the parameters of real cir-

cuits into account. The electronic design automation (EDA) community relies heavily on public benchmarks to evaluate the performance of academic and commercial design tools, because benchmarking is essential to develop effective methodologies. Hence we can analyse characteristics of benchmark circuits to provide a real picture regarding the circuit complexity. The first widely accepted benchmark suite ISCAS-85 consisting of ten industrial designs was introduced by Brglez and Fujiwara [3]. The circuits included in this benchmark set contain from 32 to 207 inputs and were originally implemented using 160–3512 logic gates. With advancements in technology, several updates have been introduced. The most recent updates are LGSynth93 [17] and IWLS 2005 [1]. LGSynth93 consists of 202 circuits with up to 24,658 gates and 1,465 registers; IWLS2005 contains 84 designs with up to 185,000 registers and 900,000 gates collected from different websites with open-source IP cores. A combinational benchmark suite was introduced by Amaru et al. [2]. This suite is designed to challenge modern logic optimization tools, and consists of 23 natively combinational circuits that are divided into three classes: arithmetic circuits, random/control circuits, and difficult benchmarks. The arithmetic benchmarks have up to 512 inputs and more than a thousand gates; the difficult benchmark set consists of circuits with more than million gates and more than 100 inputs. As is evident, a *complex circuit* is a circuit having at least a hundred inputs and consisting of thousands of gates.

The goal of this chapter is to summarize the progress in the evolutionary synthesis of gate-level digital circuits, and to identify the challenges that need to be addressed to enable evolutionary methods to penetrate into industrial practice. The rest of this chapter is organized as follows. Section 2 briefly maps the history of evolutionary circuit design. Among others, it summarizes the key results that have been achieved in the area of evolutionary design of digital circuits for the whole existence of EHW. Then, challenges identified in recent years and that should be addressed in near future are mentioned in Sect. 3. Concluding remarks are given in Sect. 4.

2 Evolutionary Design of Digital Circuits

2.1 First Generation EHW

Gate-level evolution was addressed only rarely before 2000. The first results in the area of digital circuit synthesis were reported by Koza [15], who investigated the evolutionary design of the even-parity problem in his extensive discussions of the standard genetic programming (GP) paradigm. Although the construction of an optimal parity circuit is a straightforward process, parity circuits are considered to be an appropriate benchmark problem within the evolutionary computation community when a small set of gates (AND, OR, NOT) is used.

Thompson [30] used a form of direct encoding loosely based on the structure of an FPGA in his experiment with evolution of a square wave oscillator. A genetic algorithm was employed by Coello et al. [4], to evolve various 2-bit adders and mul-

tipliers. Miller demonstrated that evolutionary design systems are not only able to rediscover standard designs, but they can, in some cases, improve them [18, 22]; he was interested in the evolutionary design of arithmetic circuits and digital filters.

A new evolutionary algorithm, Cartesian genetic programming (CGP), was introduced by Julian F. Miller in 2000.[1] Miller designed this approach to address two issues related to the efficiency of common genetic programming: poor ability to represent digital circuits, and the presence of bloat. This variant of GP is called 'Cartesian' because it represents a program using a two-dimensional grid of nodes. The genotypes are encoded as lists of integers of fixed length that are mapped to directed acyclic graphs (DAGs) rather than the usual trees. For more details about CGP and its applications, we refer the reader to the collection edited by Julian F. Miller [20].

CGP has been used to demonstrate that evolutionary computing can improve results of conventional circuit synthesis and optimization algorithms. As a proof of concept, small arithmetic circuits were considered originally. A 4-bit multiplier is the most complex circuit evolved in this category [38]. However, the problems addressed by the EHW community have remained nearly of the same complexity since then. The most complex combinational circuit directly evolved during the first two decades of EHW consists of tens of gates and has around 20 inputs [29]. It is clear that those results could barely compete with conventional circuit design tools producing circuits counting thousands of gates and hundreds of inputs.

One of the goals of the early pioneers of EHW was to evolve complex circuits, overcome the limits of traditional design, and find ways how to exploit the vast computational resources available in today's computation platforms. Unfortunately, nobody has been able to approach this goal, except Koza [16], who reports tens of human-competitive analogue circuits obtained automatically using genetic programming. Since 2000, many researchers have invested enormous effort in proposing new ways to simplify the problem for evolution in terms of finding more effective approaches to explore the search space for more complex applications. Many novel techniques, including decomposition, development, modularisation and even new representations, have been proposed [20, 27, 29, 40]. Despite this, only a little progress has been achieved and the gap between the complexity of problems addressed in industry and EHW continues to widen as the advancements in technology develop. This further supports the belief that evolutionary search works better for analogue circuits than for digital circuits, possibly due to the fact that analogue behaviours provide relatively smoother search spaces [28].

In order to address the increasing complexity of real-world designs, some authors escape from the gate-level representation and use function-level evolution. Instead of simple gates, larger building blocks such as adders and multipliers are employed. Many and even patentable digital circuits have been discovered, especially in the area of digital signal processing [20, 25, 37]. One of the most complex circuits evolved by means of the function-level approach is a random shot noise image filter with

[1] Cartesian genetic programming grew from a method of evolving digital circuits developed by [22]. However the term *Cartesian genetic programming* first appeared in [19], and was proposed as a general form of genetic programming in [21].

25 inputs consisting of more than 1,500 two-input gates when synthesised as 8-bit circuit [37]. Despite this work, EHW found itself in a critical stage around the year 2010, and it was not then clear whether there existed a path forward that would allow the field to progress [8]. The scalability problem has been identified as one of the most difficult problems that researchers are faced with in the EHW field and one that should be, among others, addressed by the second generation of EHW.

2.2 Scalability Issues

Poor scalability typically means that evolutionary algorithms are able to provide solutions to small problem instances only, and a partially working solution is usually returned in other cases. The scalability problem can primarily be seen from two perspectives: *scalability of representation* and *scalability of fitness evaluation*. From the viewpoint of the scalability of representation, the problem is that long chromosomes are usually required to represent complex solutions. Long chromosomes, however, imply large search spaces that are typically difficult to search. The scalability of fitness evaluation represents another big challenge. The problem is that complex candidate solutions might require a lot of time to be evaluated. As a consequence, only a small fraction of the search space may be explored in a reasonable time. This problem is especially noticeable in the case of the evolutionary design of digital circuits where the time required to evaluate a candidate circuit grows exponentially with the increasing number of inputs. One possibility how to overcome this issue is to reduce the number of test cases utilised to determine the fitness value. However, this approach is not applicable in the case of gate-level evolution [14]. There is a high chance that a completely different circuit violating the specification is evolved even if only a single test case is omitted.

Both mentioned scalability issues have independent effects that are hard to predict in practice. Consider, for example, the evolutionary design of gate-level multipliers from scratch. It is relatively easy to evolve a 4-bit (i.e. 8-input) multiplier, yet a huge computational effort is required to discover a 5-bit multiplier, even though the time required to evaluate a candidate solution increases only 4-fold and the number of gates is approximately doubled. Experiments with various accelerators have revealed that the number of evaluations has to be increased more than 170-fold to discover a valid 5-bit multiplier [13].

It is believed that the scalability of representation is the root cause that prevents evolutionary algorithms from handling complex instances. This hypothesis is supported mainly by the fact that the number of evaluations needed to discover a digital circuit is significantly higher compared to Koza's experiments with analogue circuits. In addition, the community has been unable to substantially improve the scalability of gate-level evolutionary design, even when various FPGA-based, GPU-based and CPU-based accelerators have been employed [13, 20, 36]. There is no doubt that the size of the search space grows enormously as the complexity of problem instances increases. On the other hand, only little is known about the phenotypic search space

of digital circuits. It may be possible, for example, that the number of valid solutions proportionally increases with the increasing size of search space.

In addition to the previously discussed scalability issues, *scalability of specification* is sometimes mentioned [35]. Let us assume that the previously mentioned problems do not exist. Even then, we will not be able to design complex circuits in reasonable time. The problem is that the frequently used specification in the form of truth table does not itself scale. The amount of memory required to store the whole truth table grows exponentially with the increasing number of variables.

2.3 Second Generation EHW

Despite pessimism of the EHW community, its researchers have continued to investigate how to overcome the scalability issues. Vasicek and Sekanina [34] demonstrate that it is feasible to optimize complex digital circuits when the common truth-table-based fitness evaluation procedure is replaced with formal verification. The method exploits the fact that efficient algorithms have been developed in the field of formal verification that enable us to relatively quickly decide whether two circuits are functionally equivalent. Compared to the state-of-the-art synthesis tools, the authors reported a 37.8% reduction in the number of gates (on average) across various benchmark circuits having 67–1408 gates and 22–128 inputs.

An interesting feature of the approach is that it focuses solely on the improvement of fitness evaluation efficiency. In order to reduce the time needed to determine the fitness value, a method routinely used in the area of logic synthesis known as combinational equivalence checking is employed. But it is not the equivalence checking alone that enables speedups of several orders of magnitude. The approach benefits from a tight connection between the evolutionary algorithm and combinational equivalence checking. Since every fitness evaluation is preceded by a mutation, a list of nodes that are different for the parent and its offspring can be calculated. This list can be used to determine a cone of influence (COI)—the set of outputs and gates that have to be compared with the reference circuit—and only these outputs and gates are checked. In addition to that, a parental circuit serves simultaneously as a reference.

The efficiency of the proposed method has been further improved by Vasicek [33], who combines a circuit simulator with formal verification in order to detect the functional non-equivalence of the parent and its offspring. In contrast with previously published work, an extensive set of 100 real-world benchmarks circuits is used to evaluate the performance of the method. The least complex circuit has 106 gates, 15 primary inputs, and 38 outputs. The most complex circuit, an audio codec controller, has 16,158 gates, 2,176 inputs, and 2,136 outputs. Half of the benchmark circuits have more than 50 primary inputs and more than a thousand gates. On average, the method enables a 34% reduction in gate count, and the evolution is executed for only 15 min.

A very pessimistic future for EHW-based digital circuit synthesis was predicted by Greenwood and Tyrrell [7]. Despite that, five years later and after little more than

15 years of EHW, Vasicek and Sekanina [34] proposed an approach easily overcoming the previous empirical limitation of evolutionary design represented by a digital circuit having about twenty inputs and up to hundred gates. At the same time, several doubts and questions emerged. Since only the scalability of fitness evaluation was addressed, it is questionable whether the scalability of representation is really the key issue preventing the handling of complex instances. We believe that this is not the case for at least the evolutionary optimization of digital circuits. Hence, addressing the problem of scalability of representation in the context of evolutionary optimization of digital circuits seems not to be urgent. The extensive experimental evaluation presented by Vasicek [33] has revealed another and probably more severe issue of evolutionary optimization of digital circuits: the enormous inefficiency of the evolutionary algorithm itself. The ratio between the number of acceptable candidate solutions and invalid candidate solutions is worse than 1:180 in average. A candidate solution is invalid if it is functionally incorrect, i.e. violates the requirements given by the specification. So approximately 99.5% of the runtime is wasted by generating and evaluating invalid candidate circuits that lie beyond the desired space of potential solutions.

If we want to keep the EHW field progressing, it seems that the researchers should firstly turn their attention away from tweaking the parameters of various evolutionary algorithms, and begin to combine evolutionary approaches with state-of-the-art methods routinely used in the field that is addressed with evolution. This claim is based mainly on the fact that the results reported by Vasicek [33] were obtained by means of the standard variant of genetic programming and standard settings recommended in the literature. Although CGP is considered as one of the most efficient methods for evolutionary design and optimization of digital circuits [20], there is no guarantee that it offers the best possible convergence. It is not even clear whether the poor efficiency discussed in the previous paragraph is caused by CGP, or if it is a common problem of all evolutionary approaches. It is unfortunately nontrivial to point out the algorithm that achieves the best performance. Firstly, many authors are evaluating algorithms using artificial problems such as simple parity circuits instead of real benchmark circuits. Secondly, only evolutionary design of simple circuits has been addressed. Hence, the efficiency of the evolutionary algorithm and its improvement represents an open question for future investigations.

The approach of [34] has sometimes been criticised that it has to be seeded with an already working circuit. and thet it cannot be used to evolve digital circuits from scratch. At first sight, this objection seems to be legitimate. However, it is important to realise and accept how circuits are specified in conventional tools. Conventional circuit synthesis and optimization usually starts from an unoptimized fully functional circuit supplied by a designer. Such a circuit is specified in the form of a netlist, typically using low-level (e.g. BLIF specification) or high-level (e.g. Verilog or VHDL language) hierarchical description. In some cases, truth tables are directly used, but to specify only simple circuits. So the seed required by evolution can easily be obtained from the specification, for example by transforming the Sum-of-Products representation into a logic network or synthesizing Verilog description to a gate-level netlist.

An interesting question, at least from the theoretical point of view, is whether complex digital circuits can be evolved from randomly seeded initial populations, i.e. whether a truly evolutionary circuit design is possible for complex circuits. Firstly, it is necessary to accept the fact that this approach can hardly compete with conventional state-of-the-art synthesis tools if the time of synthesis is considered. Obtaining a fully functional solution from a randomly seeded population consumes a considerable time because the approach exploits the generate-and-test principle, and no additional knowledge about the problem is available. However, there are reasons why it makes sense to develop the aforementioned evolutionary approach. Firstly, the circuit design problem can serve as a useful test problem for the performance evaluation and comparison of genetic programming systems. Secondly, the approach seems to be suitable for adaptive embedded systems, because running standard circuit design packages is usually infeasible on such systems.

In order to address the evolution of complex circuits from scratch, Vasicek and Sekanina [35] propose a method that exploits the fact that the specification can be given in the form of binary decision diagram (BDD). Although there are some pathological cases of circuits for which BDDs do not scale well, BDDs are known to be an efficient tool for representation and manipulation with digital circuits. Vasicek and Sekanina [35] use BDDs to determine Hamming distance between a candidate circuit and specification. This requires only a few milliseconds, even for circuits having more than 40 inputs. Compared to the well-optimized common CGP, the proposed method achieves a speedup of four to seven orders of magnitude when circuits having from 23 to 41 inputs are considered. As a proof of concept, 16 nontrivial circuits outside of the scope of well-optimized standard CGP are considered. Correctly evolved circuits are reported in twelve cases. In addition, evolution is able to improve the results of conventional synthesis tools. For example, evolution discovered a 28-input circuit having 57% fewer gates than the result obtained from the state-of-the-art synthesis tool. An average gate reduction of 48.7% is reported for all evolved circuits.

We can conclude that it has been experimentally confirmed that it is beneficial to employ EAs even in the evolution of digital circuits from scratch. In many cases, more compact solutions may be obtained, because the EA is not biased to the initial circuits that must be provided in the case of conventional synthesis. However, we have to accept and tolerate the overhead of evolution manifesting itself in the higher demand for computational resources. For example, at most three hours are required to evolve the circuits investigated by Vasicek and Sekanina [35]. On the other hand, this represents an unimportant problem when a theoretical point of view is considered. There are more fundamental questions that we should deal with.

Firstly, how is it possible that evolution successfully discovered a 28-input circuit (frg1), yet no solution was provided for 23-input circuit (cordic)? Since the 23-input circuit consists of fewer gates than the 28-input circuit, this issue is evidently not connected with poor scalability of representation. We can exclude even the problem of scalability of fitness evaluation, because the time required to evaluate the candidate solutions is nearly the same in both cases. Hence, there must be some other issue that prevents the discovery of some circuits. Secondly, the notion of circuit complexity remains an open question. Computation effort (derived from the num-

ber of input combinations and the number of gates) is used to measure the circuit complexity. Some of the circuits that are not evolved, however, exhibit a lower complexity value compared to the successfully evolved circuit. Thirdly, the paper does not offer an answer to the question related to the most complex circuit that could be directly evolved. The work was intended as a proof-of-concept only. Finally, it is worth noting that only the scalability of fitness evaluation is targeted. Does it mean that the scalability of representation is an unimportant problem, at least to some circuit size? [9] state that it is becoming generally accepted that there needs to be a move away from a direct, or one-to-one, genotype-phenotype mapping to enable evolution of large complex circuits. This conjecture seems to be wrong at least for the evolutionary optimisation of existing gate-level circuits, because large genotypes encoding circuits with thousands of gates can be directly optimised using CGP [33].

3 Open Challenges

Significant progress has been made in the area of evolutionary synthesis of digital circuits in recent years. Despite that, we are still facing many challenges that need to be properly addressed in order to enable evolutionary circuit design to become a competitive and respected synthesis method.

3.1 Evolutionary Synthesis and Hardware Community

One of the problems with the EHW community is that the practical point of view is usually in second place. The typical goal is to evolve some circuit and nobody questions whether there is a real need to do so. Although there is no doubt that much work has been done in the last more than two decades of EHW, there is a barrier preventing EHW being fully accepted by hardware community. Let us mention the main aspects that are usually criticised by the "traditional" circuit design community.

The evolutionary design of digital circuits is sometimes criticised due to its inherently non-deterministic nature. To understand this, it is necessary to realize how the conventional synthesis tools are implemented. The tools typically start with a netlist represented using a network, for example in the form of and-inverter graph (AIG). Then, deterministic transformations are performed over the AIG. Since the transformations are local in nature, the network may be refined by their repeated application. The solution quality can be improved in this way at the expense of run-time. Designers have a guarantee that the optimisation process ends. In addition, they have a certain degree of guarantee that a circuit of some quality will be obtained when, say, one hundred iterations are employed.

Another handicap of the evolutionary approaches is that they are time consuming and generally not scalable when compared with methods of logic synthesis. While the problem of scalability of evaluation has been partly eliminated, the method still

suffers from long execution times due to the inefficiency of the generate-and-test approach. Even though the recent results indicate that more complex circuits may be evolved, it will probably be necessary to accompany the proposed methods with a suitable partitioning in order to reduce the problem complexity. It makes no sense to optimize complex netlists describing the whole processors or other complex circuits at the gate-level. The conventional tools usually benefit from a hierarchical description of a given digital circuit.

Another issue is the relevance of the chosen optimisation criteria for industrial practice. Originally, the number of gates was the only criterion that was optimized within the evolutionary community. In many cases, not only the number of gates, but also delay and area on a chip, play an important role. In addition, power efficiency has emerged as one of the most critical goals of hardware design in recent years [10]. Logic synthesis is a complex process that has to consider several aspects that are in principle mutually dependent. Two basic scenarios are typically conducted in practice: optimizing the power and/or area under some delay constraints, or optimizing the delay possibly under some power and/or area constraints. In order to efficiently determine the delay, area and power dissipation of a given circuit, so as to avoid running of time-consuming SPICE simulator and yet obtain results exhibiting a reasonable accuracy, several algorithms have been developed to estimate the delay as well as power dissipation of digital circuits [10]. The evolutionary community should adopt them and combine them with evolution.

Although the attitude of hardware community to evolutionary techniques seems to be a rather sceptical, it does not mean that evolutionary synthesis has no chance to be accepted at hardware conferences. Interestingly, there is a new research area—approximate computing—in which evolutionary approaches seem to be accepted [26]. Conventional synthesis tools are not constructed to perform the synthesis of approximate circuits, and no golden design exists for an approximate circuit. In the case of approximate synthesis, it is sufficient to design a circuit that responds correctly for a reasonable subset of input vectors, provided that the worst-case (or average) error is under control. Because of the nature of approximate circuits (in fact, partially working circuits are sought) and principles of evolutionary circuit design (evolutionary-based improving of partially working circuits), evolutionary computing seems to be the approach of the first choice.

3.2 Efficiency of Cartesian Genetic Programming

Some problems, such as adopting relevant optimisation criteria, are a matter of time, but there are fundamental issues that need to be addressed. Since its introduction, CGP has been considered to be one of the most efficient methods for evolutionary design and optimization of digital circuits. However, the experiments with complex Boolean problems revealed that the evolutionary mechanisms of CGP should be revised, because there are cases for which CGP performs poorly.

It seems that the evolutionary *optimization* of digital circuits exhibits a completely different behaviour when compared to the evolutionary *design* of digital circuits. Let us give one example supporting this claim. Redundancy is believed to play an important role in evolutionary computation, as it represents a powerful mechanism for searching the fitness landscape. Two forms of redundancy have been identified in CGP [32]. Firstly, CGP employs a genotype-phenotype mapping that removes the genes coding the nodes that are inactive, i.e. the nodes that do not participate in creating the output values. This form of redundancy is referred to as *explicit genetic redundancy*. Secondly, there is *implicit genetic redundancy*. This refers to the situation when there active genes, ones that are decoded into the phenotype, that do not contribute to the behavior of the phenotype under common fitness functions that measure the distance between a candidate circuit and its specification. Both forms of genetic redundancy, together with the replacement strategy, encourages genetic drift, as individuals can be mutated without affecting their fitness. The replacement strategy in CGP enables these individuals to replace the old parent. It is believed that genetic drift strongly aids the escape from local optima. Recently, the theory that the effectiveness of evolutionary search is correlated with the number of available nodes has been disproved [32]. Despite that, it has been shown that explicit genetic redundancy offers a significant advantage. Vasicek [33] reports extensive experimental evaluation that reveals, however, that neutral mutations and redundancy surprisingly offer no advantage when using CGP to optimise complex digital circuits. This result does not mean that neutral mutations are of no significance generally, but they are less important in the case of circuit optimisation. Hence, the role of neutrality and redundancy in CGP represents still an open problem that it is not well understood, especially in the context of evolutionary synthesis of complex circuits.

CGP was originally designed to evolve digital circuits represented using a two-dimensional array of nodes. An open question is whether there is an evidence that this representation improves the efficiency of the evolutionary algorithm. Nowadays, this limitation seems to be an unnecessary constraint [32] and many researchers prefer to represent the circuits by means of a linear array of nodes. While both arrangements have the ability to represent any DAG, the main advantage of the linear encoding is that it substantially reduces the number of nodes required to encode a circuit, because the number of columns and rows of the two-dimensional array must respect circuit parameters such as the circuit depth and width. On the other hand, the two-dimensional array of nodes has the ability to explicitly limit the circuit depth. DAGs are encoded in a linear genome of integer values in CGP. Each node in the DAG is represented by a tuple of genes. One gene specifies the function that the node applies to its inputs, and the remaining genes encode where the node takes inputs from. Nodes can take input from primary circuit inputs or any node preceding them in the linear genome. This restriction prevents the creation of cycles and offers several advantages compared to the genetic programming (e.g. it allows CGP to reuse sub-circuits), but it introduces a positional bias that has a negative impact on CGP's ability to evolve certain DAGs [5]. Nodes that could be connected without creating a cycle may still be prevented from forming that connection because of the given genome ordering. This effect has been investigated only on the evolutionary design

of few small circuits, and it will be necessary to perform an additional study focused on the evolutionary optimization. Vasicek [33] did not register any significant degradation in performance of the evolutionary optimisation process related to the positional bias during the experiments with complex circuits. The manifestation of CGP's positional bias thus remains in this context an open question. Goldman and Punch [5] show that this bias is connected with an inherent pressure in CGP that makes it very difficult to evolve individuals with a higher number of active nodes. Such a situation, called the length bias, is prevalent as the length of an individual increases. We have not explicitly investigated this problem in the context of evolutionary design of complex circuits, however, this conclusion seems to be valid. The most complex circuit that has been evolved using BDDs consists of around 155 gates [35]. Fortunately, the length bias is related to evolutionary design only, and does not affect evolutionary optimization.

It is natural to expect that the previously mentioned issues can degrade the performance of CGP in some way. We are convinced, however, that the most critical part of CGP is the process of generating candidate solutions. In order to generate a new population, CGP utilizes a single genetic operator, point mutation. This operator modifies a certain number of randomly chosen genes in such a way that the value of a selected gene is replaced with a randomly generated, yet valid, new one. The range of valid values depends on the gene position and can be determined in advance. This scheme seems to be extremely inefficient. Let us give two facts supporting our claim. Firstly, a population containing only two individuals (i.e. parent and a candidate solution) provides the best convergence when a fixed number of evaluations is considered [34]. Secondly, Vasicek [33] observes that approximately 180 candidate solutions need to be generated to obtain a single valid candidate solution, i.e. an individual with the same or better fitness. Both observations suggest that the evolutionary-based approach requires the generation of a large number of candidate solutions to compensate the poor performance of the mutation operator. The poor ability to generate an acceptable candidate circuit is emphasised by the fact that the smallest possible population size exhibits the best performance. Assuming that our claims are correct, we could also explain why we have pushed forward the limits of evolution when we focus only on the improvement of the scalability of fitness evaluation. The reason is that formal methods such as BDD and SAT achieve speedup of a few orders of magnitude compared to standard CGP. We believe that this part of CGP offers a great potential to significantly improve not only the convergence but also scalability of evolutionary synthesis. It is clear that a form of informed mutations benefiting from problem domain knowledge should be employed, instead of blind random changes. The question is how we should accomplish that.

3.3 Deceptive Fitness Landscape

The notion of a fitness landscape has become an important concept in evolutionary computation. It is well known that the nature of a fitness landscape has a strong

relationship with the effectiveness of the evolutionary search. There are three aspects forming the structure of the fitness landscape: the encoding scheme, the fitness function, and the connectivity determined by the genetic operators. The structure can be specified in terms of two characteristics: ruggedness and neutrality of landscapes. Ideally, small gene changes should lead to small changes in the phenotype and, consequently, a small change in fitness value. A landscape exhibiting such a behaviour is said to be smooth. In contrast, if small movements result in large changes in fitness, then the landscape is considered rugged.

[39] study the structure of the fitness landscape by evolving 2-bit multipliers represented at the gate level. The landscape is modelled as a composition of three sub-landscapes as discussed above. The experiments reveal not only that the landscapes are irregular but also that their ruggedness is very different. The conclusion is that the fitness landscapes of digital circuits are quite challenging for evolutionary search. Unfortunately, only a little is known about the phenotypic search space of digital circuits, despite the fact that every search algorithm provides implicit assumptions about the search space [8].

If we ignore the problematic scalability issue, it is fair to say that CGP performs very well in general. However, our experiments with evolution of circuits having more than 15 inputs reveals that there are instances that are extremely hard to evolve. What is even worse, there are trivial problems for which the evolutionary search completely fails. One of them is the well-known and well-studied evolutionary design of parity circuits. In the case of the 30-input parity circuit, for example, the evolutionary search is stuck, and no correct circuit is evolved, provided that a complete set of gates (including XOR gate) is employed. To understand this phenomenon, the fitness landscape needs to be investigated. Under common conditions, there is nearly 100% probability that a randomly generated individual receives a high fitness score whose value equals $HD(F) = 2^{29}$ (i.e. 50% of the worst-case Hamming distance between the correct solution and individual capturing a Boolean function F). Among many others, constant value (i.e. $F = 0$), identities (i.e. $F = x_i$), or XOR of two input variables (i.e. $F = x_i \oplus x_j$) represent Boolean functions exhibiting $HD(F) = 2^{29}$. Even XOR over 29 input variables (i.e. $F = x_1 \oplus \ldots \oplus x_{29}$) has the same Hamming distance. In fact, every Boolean function F implemented as k-input XOR ($k = 2, \ldots, 29$) exhibits $HD(F) = 2^{29}$. Considering phenotypic space, there is a huge disproportion between the number of Boolean functions with Hamming distance equal to 2^{29} and the rest. Hence there is a high chance that such a candidate solution is produced by mutation. In addition to that, it is necessary to consider also the complexity of the correct and a partially working solution. While the correct parity circuit consists of 29 XOR gates, a circuit with Hamming distance equal to 1, for example, consists of a significantly larger number of gates.[2] It is then no surprise that it is extremely hard to generate a Boolean function consisting of more gates yet having better fitness. It

[2]Such a circuit can be implemented using a 2-input multiplexer that provides the correct output value for all input combinations except of a single combination for which the inverse output value is given. The implementation consisting of two-input gates evidently requires more gates than common parity circuit.

can be argued that evolution is an incremental process utilising neutral genetic drift that can help build a circuit consisting of more gates. Of course, but there is no pressure that helps to escape from this trap. If we analyse the number of gates during evolution, it oscillates around a few gates only.

One of the possibilities for how to eliminate this problem is to disable XOR gates. The presence of XOR gates in the function set, however, represents the main benefit for the evolutionary approach. While the conventional circuit synthesis approaches are not fully capable to perform the XOR decomposition, evolution is surprisingly able to do that very well. This is typically the main reason why such a huge reduction in the gate count is achieved in comparison with state-of-the-art synthesis tools [34]. Hence, it seems to be more beneficial to combine the Hamming distance with some additional metric (e.g. the number of used variables) to encourage the selection pressure and smooth the fitness landscape.

Evidently, a fitness function based solely on the Hamming distance produces a really deceptive fitness landscape. The question is whether parity represents a singularity or whether there exist a whole class of problems with similar behaviour. It is known that parity is a typical example of symmetric Boolean functions. The symmetric functions are Boolean functions whose output value does not depend on the permutation of the input variables. It means that the output depends only on the number of ones in the input. A unique feature of this class of Boolean functions is that a more compact representation can be utilized instead of the truth table having 2^n rows. Each symmetric Boolean function with n inputs can be represented using the $(n+1)$-tuple, whose i-th entry $(i = 0, \ldots, n)$ is the value of the function on an input vector with i ones.

4 Final Remarks

When Julian F. Miller, the (co)inventor of Cartesian genetic programming, published his paper devoted to the evolutionary design of various gate-level circuits such as adders and multipliers where he demonstrated the strength of evolutionary approach [23][3], he not only motivated researchers around the world to further develop their investigations of evolutionary design of digital circuits, but also inspired many others to engage in a relative young and promising research area: evolvable hardware. Julian brought the evolvable hardware community an efficient method for modelling and evolution of digital circuits. Despite the fact that CGP has been evaluated on small problem instances only and there is no guarantee that it will work on different problems, CGP has become very popular in EHW. Since its introduction, CGP is still considered to be one of the most efficient methods for evolutionary design and optimization of digital circuits.

[3]This paper is currently the most cited paper in the journal of Genetic Programming and Evolvable Machines.

We can say without any exaggeration that Julian gave rise to a problem that seems to be an endless source of research opportunities, challenges and questions. After little more than 15 years of CGP's existence, there exist many open questions that are still waiting to be addressed. While there are questions, such as the role of neutrality and importance of redundancy, that accompany CGP since its introduction, progress in various areas, such as evolutionary synthesis of logic circuits, has gradually revealed further questions that were not properly addressed in the past. Many problems have emerged within the last five years, during the experiments with evolutionary synthesis of complex gate-level circuits. The primary reason is that many researchers dealt only with small circuit instances in the past. Also, circuit optimization was originally out of the main focus of EHW community, because the researchers addressed the problem of circuit design.

Cartesian genetic programming, and its variants, is not the only contribution of Julian F. Miller. He is also a pioneer of an unconventional computing paradigm known as evolution-in-materio [24]. Julian is still providing us with revolutionary and inspiring ideas.

Acknowledgements This work was supported by The Ministry of Education, Youth and Sports of the Czech Republic from the National Programme of Sustainability (NPU II); project IT4Innovations excellence in science - LQ1602.

References

1. Albrecht, C.: IWLS 2005 benchmarks. Technical Report, published at 2005 International Workshop on Logic Synthesis (June 2005)
2. Amaru, L., Gaillardon, P.-E., De Micheli, G.: The EPFL combinational benchmark suite. In: 24th International Workshop on Logic & Synthesis (2015)
3. Brglez, F., Fujiwara, H.: A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran. In: Proceedings of IEEE International Symposium Circuits and Systems (ISCAS 85), pp. 677–692. IEEE Press, Piscataway, N.J. (1985)
4. Coello, C.A.C., Christiansen, A.D., Aguirre, A.H.: Automated design of combinational logic circuits by genetic algorithms. In: Artificial Neural Nets and Genetic Algorithms: Proceedings of the International Conference in Norwich, U.K., 1997, pp. 333–336. Springer, Vienna (1998). doi:10.1007/978-3-7091-6492-1_73. ISBN 978-3-7091-6492-1
5. Goldman, B.W., Punch, W.F.: Analysis of Cartesian genetic programming's evolutionary mechanisms. IEEE Trans. Evol. Comput. **19**(3):359–373 (2015). doi:10.1109/TEVC.2014.2324539. ISSN 1089-778X
6. Gordon, T.G.W., Bentley, P.J.: On evolvable hardware. In: Soft Computing in Industrial Electronics, pp. 279–323. Physica-Verlag, London, UK (2002)
7. Greenwood, G.W., Tyrrell. A.M.: Introduction to evolvable hardware: a practical guide for designing self-adaptive systems. In: IEEE Press Series on Computational Intelligence. Wiley-IEEE Press (2006). ISBN 0471719773
8. Haddow, P.C., Tyrrell, A.: Challenges of evolvable hardware: past, present and the path to a promising future. Genet. Progr. Evol. Mach. **12**, 183–215 (2011)
9. Haddow, P.C., Tufte, G., van Remortel, P.: Shrinking the genotype: L-systems for EHW? In: Proceedings of the 4th International Conference on Evolvable Systems: From Biology to Hardware. LNCS, vol. 2210, pp. 128–139. Springer (2001)

10. Hassoun, S., Sasao, T. (eds.): Logic Synthesis and Verification. Kluwer Academic Publishers, Norwell, MA, USA (2002)
11. Higuchi, T., Niwa, T., Tanaka, T., Iba, H., de Garis, H., Furuya, T.: Evolving hardware with genetic learning: a first step towards building a Darwin machine. In: Proceedings of the 2nd International Conference on Simulated Adaptive Behaviour, pp. 417–424. MIT Press (1993)
12. Higuchi, T., Liu, Y., Yao, X. (eds.): Evolvable Hardware. Springer Science+Media LLC, New York (2006)
13. Hrbacek, R., Sekanina, L.: Towards highly optimized Cartesian genetic programming: from sequential via SIMD and thread to massive parallel implementation. In: Proceedings of 2014 Annual Conference on Genetic and Evolutionary Computation, pp. 1015–1022. ACM, New York, NY, USA (2014). doi:10.1145/2576768.2598343
14. Imamura, K., Foster, J.A., Krings, A.W.: The test vector problem and limitations to evolving digital circuits. In: Proceedings of the 2nd NASA/DoD Workshop on Evolvable Hardware, pp. 75–79. IEEE Computer Society Press (2000)
15. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA (1992)
16. Koza, J.R.: Human-competitive results produced by genetic programming. Genet. Progr. Evol. Mach. **11**(3–4), 251–284 (2010)
17. McElvain, K.: LGSynth93 benchmark set version 4.0 (1993)
18. Miller, J.F.: Digital filter design at gate-level using evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 1999, pp. 1127–1134. Morgan Kaufmann (1999a)
19. Miller, J.F.: An empirical study of the efficiency of learning Boolean functions using a Cartesian Genetic Programming approach. In Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E. (Eds), Proceedings of the Genetic and Evolutionary Computation Conference, vol. 2, pp. 1135–1142. Morgan Kaufmann, Orlando, Florida, USA. 13–17 July (1999b). ISBN 1-55860-611-4
20. Miller, J.F.: Cartesian Genetic Programming, Springer (2011)
21. Miller, J.F., Thomson, P.: Cartesian genetic programming. In: Proceedings of the 3rd European Conference on Genetic Programming EuroGP2000. LNCS, vol. 1802, pp. 121–132. Springer (2000)
22. Miller, J.F., Thomson, P., Fogarty, T.: Designing electronic circuits using evolutionary algorithms. arithmetic circuits: a case study. In: Genetic Algorithms and Evolution Strategies in Engineering and Computer Science, pp. 105–131. Wiley (1997)
23. Miller, J.F., Job, D., Vassilev, V.K.: Principles in the evolutionary design of digital circuits—part I. Genet. Progr. Evol. Mach. **1**(1), 8–35 (2000)
24. Miller, J.F., Harding, S.L., Tufte, G.: Evolution-in-materio: evolving computation in materials. Evol. Intell. **7**(1):49–67 (2014). doi:10.1007/s12065-014-0106-6. ISSN 1864-5917
25. Sekanina, L.: Evolvable components: from theory to hardware implementations. In: Natural Computing Series. Springer (2004)
26. Sekanina, L., Vasicek, Z.: Evolutionary computing in approximate circuit design and optimization. In: 1st Workshop on Approximate Computing (WAPCO 2015), pp. 1–6 (2015)
27. Shanthi, A.P., Parthasarathi, R.: Practical and scalable evolution of digital circuits. Appl. Soft Comput. **9**(2), 618–624 (2009)
28. Stoica, A., Keymeulen, D., Tawel, R., Salazar-Lazaro, C., Li, W.-T.: Evolutionary experiments with a fine-grained reconfigurable architecture for analog and digital CMOS circuits. In: Proceedings of the 1st NASA/DOD workshop on Evolvable Hardware, EH 1999, pp. 76–84. IEEE Computer Society, Washington, DC, USA (1999)
29. Stomeo, E., Kalganova, T., Lambert, C.: Generalized disjunction decomposition for evolvable hardware. IEEE Trans. Syst. Man Cybern. Part B **36**(5), 1024–1043 (2006)
30. Thompson, A.: Silicon evolution. In: Proceedings of the First Annual Conference on Genetic Programming, GECCO '96, pp. 444–452. MIT Press, Cambridge, MA, USA (1996)
31. Trefzer, M.A., Tyrrell, A.M.: Evolvable Hardware: From Practice to Application. Springer, Berlin, Heidelberg (2015)

32. Turner, A.J., Miller, J.F.: Neutral genetic drift: an investigation using Cartesian genetic programming. Genet. Progr. Evol. Mach. **16**(4):531–558 (2015). doi:10.1007/s10710-015-9244-6. ISSN 1573-7632
33. Vasicek, Z.: Cartesian GP in optimization of combinational circuits with hundreds of inputs and thousands of gates. In: Proceedings of the 18th European Conference on Genetic Programming—EuroGP. LCNS 9025, pp. 139–150. Springer International Publishing (2015)
34. Vasicek, Z., Sekanina, L.: Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. Genet. Progr. Evol. Mach. **12**(3), 305–327 (2011)
35. Vasicek, Z., Sekanina, L.: How to evolve complex combinational circuits from scratch? In: Proceedings of the 2014 IEEE International Conference on Evolvable Systems, pp. 133–140. IEEE (2014)
36. Vasicek, Z., Slany, K.: Efficient phenotype evaluation in Cartesian genetic programming. In: Proceedings of the 15th European Conference on Genetic Programming. LNCS 7244, pp. 266–278. Springer (2012)
37. Vasicek, Z., Bidlo, M., Sekanina, L.: Evolution of efficient real-time non-linear image filters for fpgas. Soft Comput. **17**(11):2163–2180 (2013). doi:10.1007/s00500-013-1040-8. ISSN 1433-7479
38. Vassilev, V., Job, D., Miller, J.F.: Towards the automatic design of more efficient digital circuits. In: Lohn, J., Stoica, A., Keymeulen, D., Colombano, S. (eds.) Proceedings of the 2nd NASA/DoD Workshop on Evolvable Hardware, pp. 151–160. IEEE Computer Society, Los Alamitos, CA, USA (2000)
39. Vassilev, V.K., Miller, J.F., Fogarty, T.C.: Digital circuit evolution and fitness landscapes. In: Proceedings of the Congress on Evolutionary Computation, vol. 2. IEEE Press, 6-9 July (1999)
40. Zhao, S., Jiao, L.: Multi-objective evolutionary design and knowledge discovery of logic circuits based on an adaptive genetic algorithm. Genet. Progr. Evol. Mach. **7**(3), 195–210 (2006)

http://www.springer.com/978-3-319-67996-9

Inspired by Nature
Essays Presented to Julian F. Miller on the Occasion of his 60th Birthday
Stepney, S.; Adamatzky, A. (Eds.)
2018, X, 387 p. 168 illus., 78 illus. in color., Hardcover
ISBN: 978-3-319-67996-9