Approximate Computing: An Old Job for Cartesian Genetic Programming?

Lukas Sekanina

Abstract Miller's Cartesian genetic programming (CGP) has significantly influenced the development of evolutionary circuit design and evolvable hardware. We present key ingredients of CGP with respect to the efficient search in the space of digital circuits. We then show that approximate computing, which is currently one of the promising approaches used to reduce power consumption of computer systems, is a natural application for CGP. We briefly survey typical applications of CGP in approximate circuit design and outline new directions in approximate computing that could benefit from CGP.

1 Introduction

Julian F. Miller, a pioneer of genetic programming (GP), evolvable hardware (EHW), evolution in materio and other approaches that could be together classified as unconventional computing paradigms, is best known as the (co)inventor of Cartesian genetic programming (CGP) [25]. Genetic programming is a computational intelligence method capable of the automated designing of programs. It has been developed since the eighties, with the significant contribution of John Koza who mainly worked on a tree-based GP in which candidate solutions are syntactic trees [20]. In Miller's version of genetic programming, CGP, candidate solutions are represented using directed acyclic graphs encoded as finite-size strings of integers. This representation is especially useful for evolving electronic circuits as it naturally captures the circuit physical structure and supports multiple outputs, subgraph sharing and various types of elementary circuit components.

CGP has predominantly been developed within two research communities: genetic programming and evolvable hardware. For the genetic programming community, interesting concepts behind CGP are the specific genotype-phenotype mapping, mutation-driven search over small populations, role of neutrality and bloat free

L. Sekanina (✉)
Faculty of Information Technology, Brno University of Technology,
IT4Innovations Centre of Excellence, Brno, Czech Republic
e-mail: sekanina@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_9

search. In the last decade, CGP was extended in various directions, for example, the evolution of modular code was supported and self-modifying CGP enabled the evolved programs to change themselves as a function of time. For the evolvable hardware community, CGP has represented a natural way to model and evolve digital circuits, not only in a circuit simulator but also directly on a chip. New circuit designs, including patented solutions,[1] were routinely obtained by means of CGP. Digital architectures capable of autonomous adaptation and self-repair were constructed, where CGP was responsible for autonomous changes of the circuits involved. CGP-based results have also been published outside the major evolutionary computing or evolvable hardware events, thus influencing people focusing on the mainstream circuit design approach.

In computer engineering, the following trends can be observed [7, 22]:

- Energy efficiency is strongly requested, especially for omnipresent wireless battery-powered systems (such as smart phones, consumer electronics and wireless sensor networks) and, on the other hand, for energy demanding data and supercomputer centers processing Big Data.
- Computationally-demanding applications (e.g. those based on deep neural networks) are implemented even in small battery powered embedded systems.
- Integrated circuits developed using recent fabrication technology exhibit reliability issues and uncertainties, especially when operated at low voltages.
- Massive parallelism available by implementing billions of transistors on a single die cannot be fully employed as only a fraction of them can be activated at the same time in order to prevent burning the chip.

One of the approaches capable of coping (at least partly) with the aforementioned issues is *approximate computing* [31]. It exploits the fact that many applications are inherently error resilient. If some errors can be tolerated, then the underlying hardware and software can be simplified to be faster or less energy consuming. Multimedia, datamining and machine learning applications are good candidates for the approximation because possible small imperfections in data processing usually have only a small impact on the quality of result. The approximate hardware and software system design can be viewed as an optimization problem in which a good trade-off is sought among key design variables—performance, energy and error. This optimization problem can be attacked with CGP [60, 61]. A real challenge is to employ approximate computing for the most advanced technology nodes, where transistors parameters exhibit unusual variability. One of the approaches is, by graceful performance degradation at run-time and in the presence of uncertainties or errors, to deliver the best possible trade-off between the quality of result and energy consumption for all components on a chip. It has to be noted that this new notion of approximation, which was established around 2010, is different from the approximations that have been conducted for decades in the fields of computer engineering (such as approximate signal processing by [38]) and computer science (such as approximate algorithms by [64]).

[1] Nonlinear image filter, Patent No. 304181, Czech Republic, 2013.

The aim of this chapter is to emphasize Miller's contribution to the evolutionary design of gate level circuits. In particular, we highlight some of the ingredients of CGP that are crucial for an efficient search, especially in the space of digital circuits (Sect. 2). We show that some of Miller's results in the area of evolutionary circuit design that were obtained over 15 years ago can be interpreted as vital contributions to the current research in approximate computing (Sect. 3). We briefly survey current applications of CGP in approximate circuit design (Sect. 4). The last part of this chapter is devoted to new directions in approximate computing that could benefit from CGP (Sect. 5). Final remarks are given in Sect. 6.

2 Cartesian Genetic Programming

According to Julian F. Miller:

> Cartesian genetic programming grew from a method of evolving digital circuits developed by Miller et al. in 1997 [29]. However the term 'Cartesian genetic programming' first appeared in 1999 [23] and was proposed as a general form of genetic programming in 2000 [28].

In CGP, candidate solutions are represented in a two-dimensional array of programmable nodes. While the phenotype is represented using a directed oriented graph, the genotype is encoded in a fixed-size string of integers. In this section, key ingredients of CGP are briefly introduced. The aim is not to provide a detailed description (which is available in [25]), but to emphasize important consequences for practical evolutionary circuit design. We primarily deal with combinational gate-level circuits.

2.1 Circuit Representation

An n_i-input and n_o-output combinational circuit is modelled using an array of $n_c \cdot n_r$ programmable nodes forming a Cartesian grid. A set of available n_a-input node functions is denoted Γ, where elementary logic functions are included for gate-level circuits. The levels-back parameter l constraints which columns a node can get its inputs from. No feedbacks are allowed in the basic version of CGP. The primary inputs and programmable nodes are uniquely numbered. For each node the chromosome contains (n_a+1) values that represent the node function and n_a addresses specifying the input connections. The chromosome also contains n_o values specifying the gates connected to the primary outputs. The chromosome size is $n_c n_r(n_a + 1) + n_o$ integers.

The circuit representation used by CGP was not invented by Miller as there are papers from the early nineties utilizing the same concept [21, 37]. An important property of this representation is its extensibility to support different types of components (such as gates, 3-electrode transistors, or 6-input look-up tables), recurrent

networks [51], transistor-level circuits [68], modular circuits [14, 67], decomposition strategies [49] and self-modifying code [8].

No restrictions on resulting circuits are given if $n_r = 1$ and $l = n_c$, which is the most common setting used in the literature. However, this setting is not suitable for an on-chip implementation of CGP as many options for interconnecting the nodes are allowed and must be supported by the underlying hardware platform. On the other hand, if $l = 1$ then pipeline circuits can naturally be evolved and implemented.

After decoding the genotype, each node is labelled as active or inactive according to its utilization in the phenotype. Regarding the size of the array, one of the conclusions in the literature is that more is better (i.e. the number of evaluations is minimal when the genome size far exceeds what is necessary for the problem at hand [27]). A recent work showed that there is a problem-dependent limit in the number of nodes that makes sense to employ in the array [52]. However, all results were obtained for relatively small problem instances (such as 4-bit multiplier or 8-bit parity requiring tens of gates to be implemented). It is an open question if this high redundancy is useful for the evolution of real-world circuits containing thousands of gates.

2.2 Genetic Operators

Despise several attempts to introduce a useful crossover operator to CGP [2, 47], mutation is the only genetic operator employed in almost all studies and applications conducted with CGP. The structure of fitness landscapes was studied and it has been shown that the landscapes are vastly neutral with sharply differentiated plateaus [30]. The search space is considered as very difficult, especially for circuits such as parallel multipliers. The recent work of Goldman and Punch analysed several types of mutation and their impact of the quality of search, stressing the importance of mutations on active over inactive genes [6]. Vasicek showed that there is no reason to support neutral mutations for evolutionary optimization of complex circuits (hundreds of inputs, thousands of gates) if the task is to minimize the number of gates in a fully functional circuit [53].

2.3 Seeding the Initial Population

If the initial population is randomly generated, we speak about *evolutionary circuit design*. The task for CGP is to provide a circuit structure. If conventional solutions are used in the initial population (i.e. the circuit structure is known), the goal of CGP is to optimize circuit parameters (as well as structure) and then we speak about *evolutionary circuit optimization*. While it is very time consuming to evolve non-trivial circuits from scratch, the question is whether it makes sense to evolve them at all. We believe that there are at least two reasons.

Firstly, the approach seems to be promising for adaptive embedded systems. If a new logic function has to be implemented in reconfigurable hardware, employing CGP could be a good choice in the case of simple logic circuits. The reason is that it is usually impossible to perform a standard circuit design and synthesis procedure by means of common circuit design software executed directly in the embedded system because of its high time and resources requirements. On the other hand, CGP, which is generating and testing candidate solutions directly in the reconfigurable device, can provide a suitable solution in a reasonable time, even if some circuit components are faulty [40]. This is impossible using conventional synthesis, placement and routing algorithms which expect fault-free chips.

Secondly, the circuit design problem can serve as a useful test problem for the performance evaluation and comparison of genetic programming systems. It is also occasionally possible to obtain new useful implementations of these circuits, unbiased with respect to conventional circuit designs. The most complex circuits evolved from scratch (without any decomposition) were reported in [58] and they contain less than 30 inputs and a thousand gates.

However, in the circuit design and optimization practice, there is no reason to start from scratch. A fully functional solution can always be generated from the specification by means of a basic conventional circuit design method. The most complex circuits optimized by CGP that was seeded with the best known conventional implementations were reported by [53]. They contained hundreds of inputs and thousands of gates.

It should be noted that the approach discussed so far assumes that a circuit fully compliant with the specification must be delivered. This scenario is referred to as the evolution of *completely specified circuits* and is primarily relevant for arithmetic circuits and control logic. On the other hand, *incompletely specified circuits* are used in applications (such as classification, filtering, hashing and prediction) in which the correctness can only be evaluated using a subset of all possible input vectors because verifying responses for all possible input combinations is intractable. According to the applications reported in the literature, it seems that the evolutionary circuit design is more successful for the incompletely specified circuits. We see in Sect. 3.1 that both approaches are relevant for approximate computing.

2.4 Search Algorithm

While the problem representation used in CGP is not an original invention of Julian F. Miller, the mutation-based search algorithm operating over this representation is fundamental in Miller's contribution. The search algorithm utilized by CGP is a simple $(1 + \lambda)$ search strategy. Every new population consists of the parent and its λ offspring created by a mutation operator. The parent is always the highest-scored candidate circuit. The parent from the previous generation is never selected as a new parent if there is another offspring with the same fitness value. This rule is crucial for an efficient search as it introduces new genetic material into the population through

the neutral mutations. The algorithm is terminated when the maximum number of generations is exhausted or a sufficiently working solution is obtained.

It was observed in [5] that if CGP is minimizing the number of gates in an already fully functional circuit, then a modified parent selection mechanism is more efficient than the standard one. Quite unintuitively, the selection of the parent individual based solely on its functionality (i.e. all fully functional offspring can then become the new parent independent of their size) instead of compactness, led to smaller phenotypes at the end of evolution.

Contrasted to the tree-based GP, CGP employs a very small population (λ is usually between 1 and 10), but many generations are produced. Introducing a multiobjective or co-evolutionary CGP is then problematic as the populations are very small. Despite this fact, several multiobjective CGP implementations [11, 13, 16] and co-evolutionary CGP implementations [46] have been proposed.

2.5 Fitness Evaluation and Its Acceleration

Since its introduction, CGP has been promoted as a universal form of GP, but is especially useful for circuit design and optimization. Many authors reported digital circuits that can be evolved from scratch or optimized by CGP and at the same time these circuits show some improvements in terms of the number of gates (and delay in some cases) against conventional implementations (one of the first results in this direction was reported by [63]). The most common approach to construct the fitness function $f()$ is as follows:

$$f = \begin{cases} b & \textit{when} \quad b < n_o 2^{n_i}, \\ b + (n_c n_r - z) & \textit{otherwise}, \end{cases} \tag{1}$$

where b is the number of correct output bits obtained as response for all possible assignments to the inputs, z denotes the number of gates utilized in a particular candidate circuit and $n_c \cdot n_r$ is the total number of available gates. It can be seen that the last term $n_c n_r - z$ is considered only if the circuit behavior is perfect

(i.e. $b = b_{max} = n_o 2^{n_i}$). We can observe that the evolution has to discover a perfectly working solution firstly, while the size of circuit is not important. Then, the number of gates is optimized.

For the incompletely specified problems, the fitness is usually calculated on the basis of candidate circuit responses for a given training data set. The evolved circuit has to be validated using a test set. This approach has been widely adopted in the evolutionary design of image filters [42], classifiers [17], hash functions [15] and other circuits.

As the fitness calculation is the most time consuming procedure of CGP, various accelerators have been proposed to reduce the circuit evaluation time. The approaches include bit-level parallel circuit simulation and fitness function precompilation [62], parallel CGP [13], and FPGA [3, 54] and GPU [9] based accelerators.

2.6 *Practical Aspects of Evolutionary Circuit Design*

CGP for logic synthesis and optimization, as discussed so far, has not been widely accepted by the circuit design community. The reason is that this approach is far from the practical needs of the community. The main criticism addressed the issues of the (1) initial solution selection (i.e. there is usually no reason to evolve a circuit from scratch, see Sect. 2.3), (2) simplified modelling of circuit parameters, (3) poor scalability of the method, (4) non-deterministic behaviour of the method and (5) validation of the method using a few benchmark circuits only. We elaborate (2) and (3) in greater detail in the following paragraphs.

The fitness function according to Eq. 1 would be acceptable for logic synthesis, where the goal is to minimize the number of gates. It is insufficient for circuits that have to be implemented on a chip, where detailed information about the area on a chip, delay and power consumption are important. In order to estimate parameters of a given circuit, a detailed circuit analysis is requested. As many candidate circuits have to be evaluated it is very time consuming to call a professional circuit simulator for each circuit. Hence parameters of candidate circuits are estimated in the fitness function and a complete measurement is performed for the best circuits at the end of evolution. This methodology was implemented, for example, in paper [33, 59], in which the area and delay were estimated using the parameters defined in the liberty timing file available for a given semiconductor technology. Delay of a gate was modelled as a function of its input transition time and the output capacitive load. The delay of the whole circuit was then determined as a delay along the longest path. The total area was calculated as the sum of areas of all gates involved in the circuit. Finally, power consumption was estimated according to the methodology introduced for gate- and transistor-level circuits in [33]. In another transistor-level approach, power consumption and other circuit parameters were computed for all candidate designs using a circuit simulator of the Spice family [69].

Coping with the scalability problems is a more serious issue. The problem is that the circuit evaluation time (Eq. 1) grows exponentially with the number of inputs.

Hence the method is only applicable to the evolution of relatively small circuits. Because the specification is given in the form of a truth table, it is impossible to specify complex circuits in practice. Several methods have been proposed to eliminate these problems (see the overview in [43]). However, the most complex circuits evolved in this scenario are, for example, 6-bit multipliers, 9-bit adders, and 17-bit parity circuits [13, 49].

A promising solution to the fitness evaluation scalability problem is based on a completely different strategy to the fitness evaluation. In practice, a common situation is that a highly unoptimized, but fully functional circuit implementation, is always available. Such a circuit can be used in the initial population of CGP. The new fitness evaluation procedure then exploits the fact that efficient algorithms, which allow us to decide relatively quickly on whether two circuits are functionally equivalent, were developed in the field of formal verification. In our context, the task is to decide whether the parent and its offspring (created by a mutation operator) are functionally equivalent, assuming that the evolutionary algorithm is seeded by a fully functional solution and that the current parent is also fully functional. If the equivalence holds, the fitness of the offspring is given by the number of gates if the task is to minimize the number of gates. An evolutionary circuit optimization method was introduced in [55], which employs a satisfiability problem solver (SAT solver) in the fitness function in order to decide the functional equivalence. An average gate reduction of 25% was reported for benchmark circuits containing thousands of gates and having tens of inputs in comparison with state of the art academia, as well as commercial tools [56]. This result was improved in [53] by using a circuit simulator prior to a SAT solver to disprove the equivalence between a candidate solution and its parent. If the equivalence checking is based on binary decision diagrams (BDD), the circuits can also be evolved from scratch because it is possible and relatively easy to obtain the Hamming distance between the outputs of the candidate circuit and the specification [58]. However, the BDD-based approach seems to be less scalable than the SAT-based fitness evaluation.

We can summarize that CGP is currently applicable in the evolutionary design and optimization of relatively complex combinational circuits. CGP can work as a multi-objective design method in which key circuit parameters are directly estimated in the fitness evaluation procedure and optimized in the course of evolution. While improving conventionally optimized circuits is the clear advantage of CGP, its huge execution time is the main drawback.

3 Approximate Computing and Evolvable Hardware

The research dealing with approximate computing has been substantially growing since 2010. We briefly introduce this concept in this section. We emphasize that some research performed in CGP and evolvable hardware community over 10 years ago is currently very relevant for approximate computing.

3.1 Approximate Computing

According to [31]:

> Approximate computing exploits the gap between the level of accuracy required by the applications/users and that provided by the computing system, for achieving diverse optimizations.

Approximate computing was established with the goal of providing more energy efficient, faster, and less complex computer-based systems by allowing some errors in computations. One of motivations for approximate computing is that the exact computing utilizing nanometer transistors provided by recent technology nodes is extremely expensive in terms of energy requirements and reliable behavior. An open question is how to effectively and reliably compute with a huge amount of unreliable components. Another motivation is that many applications (typically in the areas of multimedia, graphics, data mining, and big data processing) are inherently *error resilient*. This resilience can be exploited in such a way that the error is exchanged for improvements in power consumption, throughput or implementation cost.

One of the approximation techniques is *functional approximation* whose principle is to implement a slightly different function with respect to the original one, provided that the error is acceptable and key system parameters are improved. The functional approximation can be conducted at the level of software as well as hardware. An approximate solution is typically obtained by a heuristic procedure that modifies the original implementation. For example, artificial neural networks were used to approximate software modules [4] and search-based methods allowed us to approximate some hardware components [39].

In addition to functional approximations, timing induced approximations (voltage over scaling) and components showing "unreliable" behavior (such as approximate memory elements) are often employed. The aim is to support approximate computing at the level of programming languages [41] and exploit new features of specialized approximate processors [1].

3.2 Approximations with CGP Before the Approximate Computing Era

Two main directions in approximate computing are (1) energy-efficient computing with unreliable components that are present in the new chips, and (2) approximation of circuits and programs on conventional platforms in order to reduce energy requirements. Interestingly, both directions can be traced in Miller's research on evolvable hardware and CGP.

3.2.1 Evolution in Materio

In the mid-1990s, Adrian Thompson evolved a tone discriminator circuit directly in the XC6216 FPGA chip. The discriminator required significantly less resources than usual conventionally designed solutions would occupy in the same FPGA [50]. Despite a huge effort, Thompson has never fully understood the evolved design. The evolved discriminator was fully functional, but its robustness was limited. For example, a higher sensitivity to fluctuations in environment (external temperature, power supply voltage) and dependence on a particular piece of FPGA were reported. This result, showing an innovative trade-off between the robustness and the amount of resources in the FPGA, can be considered as an early approach to approximate circuit design by means of evolutionary algorithms.

Thompson's work was for Julian F. Miller and his collaborators a starting point in continuing the investigation of the exploitation of physical properties of suitable substrates using artificial evolution. They have developed a new concept which is currently referred to as the *evolution in materio* (i.e. evolving useful functions in a physical system without understanding the rules of the game [26]). The idea is that there is a material to which physical signals can be applied or measured via a set of electrodes. A computer controls the application of physical inputs applied to the material, reading of physical signals from the material and the application to the material of other physical inputs known as physical configurations. Successful examples include evolved logic functions in liquid crystals [10] and carbon nanotubes materials [32].

In order to study and exploit "physics" (known from the evolution in materio) in computer simulation, Miller contributed to the development of the so-called *messy gates*. A messy gate is a gate-like component with added noise. CGP was extended to support noise modelling and then used to evolve small combinational circuits composed of messy gates. Evolved circuits exhibited implicit fault tolerance. Moreover, surprisingly efficient and robust designs were obtained for small combinational circuits.

The above-mentioned studies show that reliable computing on unknown and unreliable platforms has been one of Miller's central research topics for a long time. This topic has been further explored within an EU-funded project, NASCENCE.[2]

3.2.2 Functional Approximation and Approximation of Functions

The functional approximation is frequently used to approximate digital circuits such as adders, multipliers, filters and general logic. In 1999, Miller introduced a CGP-based method for a finite impulse response (FIR) filter design [24] that would be called functional approximation nowadays. In this method, candidate filters are composed of elementary logic gates, thus ignoring completely the well-developed techniques based on multiply–and–accumulate structures. Evolved networks of gates are extremely area-efficient (and thus potentially energy efficient) in comparison with

[2]NAnoSCale Engineering for Novel Computation using Evolution, http://www.nascence.eu.

conventional filters. However, only partial functionally has been obtained because of the overall simplicity of the logic networks. The evolved circuits are not, in fact, filters. In most cases, they are combinational quasi-linear circuits trained on some data. They are not able to generalize for the input signals unseen during the evolution. In order to obtain real filters, the design process must guarantee that the evolved circuits are linear, which is not the case in this method. One of the possible benefits of the method is that circuits can be evolved to perform filtering task, even if sufficient resources are not available (e.g. a part of chip is damaged). Furthermore, as Miller noted, "The origin of the quasi-linearity is at present quite mysterious ... Currently there is no known mathematical way of designing filters directly at this level".

In mathematics, it is investigated how certain (usually complex) functions can best be approximated by means of basic functions that are inexpensive or suitable according to a given purpose, and with quantitatively characterizing the errors thereby introduced. CGP (and GP in general) is a method capable of performing the so-called *symbolic regression* (i.e. providing an expression which represents (with some error) an unknown function for a given data set). Miller was one of the core persons who established a *self-modifying* CGP (SMCGP), which can find general solutions to classes of problems and mathematical algorithms such as arbitrary parity, n-bit binary addition, sequences that compute π and e, etc. [8]. SMCGP is a developmental form of CGP that supports self-modification functions in addition to computational functions and enables phenotypes to vary over time. By means of the phenotype development, the degree of approximation of the target behaviour (and thus the corresponding error) can be tuned.

4 Circuit Approximation by Means of CGP

This section briefly surveys recent applications of CGP is which approximate computing is explicitly mentioned as the target application. The approximate circuit design problem is formulated as a *multi-objective* optimization problem in which the accuracy, area, delay (or performance) and power consumption are conflicting design objectives. The CGP-based approximation methods typically include the following attributes:

- Circuit parameters (delay, area, power consumption) are modelled more precisely than in the applications discussed in the previous sections.
- The search algorithm is typically constructed as multiobjective and seeded by conventional implementations showing different circuit parameters. It is assumed that the user will choose the most suitable trade-off from the Pareto front for a particular application.
- Different types of error metrics are employed. If a candidate circuit is evaluated using all possible input combinations, an arbitrary error metric can be computed. In the case of complex circuits, formal methods based on SAT solving or BDDs are currently applicable for evaluating only a few relevant error metrics [12].

In the following sections, three CGP-based approaches for circuit approximation are presented. Finally, CGP utilizing relaxed equivalence checking in the fitness function is discussed.

4.1 Resources-Oriented Method

This method exploits the fact that CGP can produce a partially working solution even if sufficient resources for constructing an accurate circuit are not available. The idea is to evolve a circuit showing a minimum error using k_i components (gates) provided that $k_i < K$ and K is the number of components (gates) required to implement a correct circuit. CGP is considered as a single-objective method which is executed several times with different k_i as the parameter. It provides a set of approximate circuits, each of which typically exhibits a different trade-off between the functionality and the number of gates. The main advantage is that the user can control the used area (and power consumption) more comfortably than by means of the error-oriented methods. The method was employed to approximate small multipliers and 9-input and 25-input median circuits operating over 8 bits [60]. Approximate software implementations of the median function were evaluated for microcontrollers by [34].

4.2 Error-Oriented Method

Vasicek and Sekanina [57] proposed a complementary design approach. The user is supposed to define the required error level e_{max} (e.g. the average error magnitude). In the first step, CGP, which is seeded by a conventional fully functional implementation, is utilized to modify the seed in order to obtain a circuit with a predefined e_{max}. After obtaining that circuit, in the second step, CGP can minimize the number of gates or other criteria providing that e_{max} is left unchanged. The method is again a single objective and multiple runs are required to construct the Pareto front. The method was evaluated in the task of approximate multipliers design. The error-oriented approach tends to be less computationally demanding than the resources oriented method.

Approximate multipliers showing specific properties were evolved for artificial neural networks implemented on a chip. Their utilization led to a significant power consumption reduction and only a very small loss in the accuracy of the image classification [35].

4.3 Multi-objective CGP

In the multi-objective method, the error and other key circuit parameters (area, delay and power consumption) are optimized together by an algorithm combining CGP with NSGA-II [36, 59]. For example, in [36], the initial population was seeded using

13 different conventional 8-bit adders and 6 different conventional 8-bit multipliers in the task of the adder and multiplier approximation. The gate set contained components of a 180 nm process library, including half and full adders. The mean relative error was used in the fitness function while the predefined worst case error and worst case relative error constrained the search space. As the task is very computationally demanding, a highly parallel implementation of CGP, exploiting a vectorised and multi-threaded code, was employed. This approach enabled us to evolve a rich library of adders and multipliers (showing different errors and parameters) that can be utilized in future applications of approximate computing.

4.4 Relaxed Equivalence Checking

So far we have discussed the approximation methods that evaluate the candidate solutions by applying a set of (all) input vectors and measuring the error of the output vectors with respect to an exact solution. This approach is not, however, applicable when approximating complex circuits. If the exact error of the approximation has to be determined, formal *relaxed* equivalence checking is requested, stressing the fact that the considered systems will be checked to be equal up to some bound w.r.t., a suitably chosen distance metric. This research area is rather unexplored as almost all formal approaches have been developed for (exact) equivalence checking [12]. Checking the worst error can be based on satisfiability (SAT) solving as demonstrated in [66]. However, while violating the worst error can be detected, no efficient method capable of establishing, for example, the average error using a SAT solver has been proposed up to now. Approaches based on binary decision diagrams (BDDs) in order to determine the average arithmetic error, worst error, and error rate were introduced [48].

In the context of CGP, a fitness function based on the average Hamming distance computed by means of BDD was introduced by [61]. The method enabled us to approximate combinational circuits consisting of hundreds of gates and up to 50 inputs (i.e. the circuits whose evolution and approximation is impossible with a direct implementation of the fitness function defined in Eq. 1).

5 New Directions

Although evolutionary approximation of arithmetic circuits [36] and general logic [61] is possible with CGP, and CGP seems to be a quite competitive method, there are other promising areas for CGP in approximate computing. If a candidate

approximate circuit is evaluated using a data set (e.g. in applications such as image filtering, classification and prediction), then the scalability issues of CGP are not as burning as in the case of arithmetic circuits. Hence, we expect that CGP would be very successful in these domains.

5.1 Quality Configurable Circuits

In applications such as signal processing or image compression, it is useful when the quality of approximation (and a corresponding circuit error) can dynamically be adapted in "situ" as a response to variable requirements on the quality of results and available resources. This adaptive quality control is addressed by methods recently developed in the area of approximate computing [45, 65].

This concept has, however, been explored by the evolvable hardware community in the past. For example, a polymorphic FIR filter was proposed which can operate in two modes: a standard one and one with a reduced power budget. When the filter operates with a reduced power budget, some filter coefficients are reconfigured and some parts of the filter are disconnected. The goal is to reconfigure the filter in such a way that its original function is approximated as precisely as possible. The reconfiguration of coefficients is implemented using multiplierless polymorphic constant multipliers whose implementation was evolved using CGP [44]. In another study, Kneiper et al. investigated the robustness of EHW-based classifiers that are able to cope with changing resources at run-time. The performance and accuracy was recognized as sufficient as long as a certain amount of resources are present in the system [19]. It seems to be a natural task for CGP to automatically divide a target circuit into several sections and optimize these sections in such a way that a quality configurable solution is obtained.

5.2 Approximate Neural Networks

CGP has already been used in order to approximate neural networks, in particular, the multipliers involved in computing the product of the inputs and synaptic weights were approximated by [35]. However, there is a new opportunity for CGP in approximate neural networks. Miller has managed CGP to encode and evolve artificial neural networks. CGP is able to simultaneously evolve the networks connections weights, topology and neuron transfer functions [18]. It is also compatible with recurrent-CGP enabling the evolution of recurrent neural networks. It seems straightforward to evolve approximate neural network implementations by means of CGP and find a good trade-off between the network error and power consumption.

6 Final Remarks

In this chapter, we highlighted Julian F. Miller's contribution to the evolutionary circuit design and identified a strong connection and relevance of his research to approximate computing.

It is fair to say that this chapter could be seen as a bit biased as many of the results and studies referenced were developed by the Evolvable hardware group at Faculty of Information Technology, Brno University of Technology. However, it indicates how strongly our work was influenced by the concepts and methods proposed by Julian F. Miller. We hope that Miller's ideas were exploited by us in a good way and that we helped to disseminate these ideas to other communities, outside the evolutionary computing (Miller's home), such as mainstream circuit design conferences (DATE, ICCAD, FPL and DDECS).

Thank you, Julian!

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. Chippa, V., Venkataramani, S., Chakradhar, S., Roy, K., Raghunathan, A.: Approximate computing: an integrated hardware approach. In: 2013 Asilomar Conference on Signals, Systems and Computers, pp. 111–117. IEEE (2013)
2. Clegg, J., Walker, J.A., Miller, J.F.: A new crossover technique for cartesian genetic programming. In: Proceedings of GECCO, pp. 1580–1587. ACM (2007)
3. Dobai, R., Sekanina, L.: Low-level flexible architecture with hybrid reconfiguration for evolvable hardware. ACM Trans. Reconfig. Technol. Syst. **8**(3), 1–24 (2015)
4. Esmaeilzadeh, H., Sampson, A., Ceze, L., Burger, D.: Neural acceleration for general-purpose approximate programs. Commun. ACM **58**(1), 105–115 (2015)
5. Gajda, Z., Sekanina, L.: An efficient selection strategy for digital circuit evolution. In: Evolvable Systems: From Biology to Hardware, LNCS, vol. 6274, pp. 13–24. Springer (2010)
6. Goldman, B.W., Punch, W.F.: Analysis of Cartesian genetic programming's evolutionary mechanisms. IEEE Trans. Evol. Comput. **19**(3), 359–373 (2015)
7. Gupta, P., Agarwal, Y., Dolecek, L., Dutt, N., Gupta, R.K., Kumar, R., Mitra, S., Nicolau, A., Rosing, T.S., Srivastava, M.B., Swanson, S., Sylvester, D.: Underdesigned and opportunistic computing in presence of hardware variability. IEEE Trans. CAD Integr. Circuits Syst. **32**(1), 8–23 (2013)
8. Harding, S., Miller, J.F., Banzhaf, W.: Developments in cartesian genetic programming: self-modifying CGP. Genet. Program. Evolvable Mach. **11**(3–4), 397–439 (2010)
9. Harding, S.L., Banzhaf, W.: Hardware acceleration for CGP: graphics processing units. In: Cartesian Genetic Programming, pp. 231–253. Springer (2011)
10. Harding, S.L., Miller, J.F.: Evolution in materio: evolving logic gates in liquid crystal. Int. J. Unconv. Comput. **3**(4), 243–257 (2007)
11. Hilder, J., Walker, J., Tyrrell, A.: Use of a multi-objective fitness function to improve cartesian genetic programming circuits. In: NASA/ESA Conference on Adaptive Hardware and Systems, pp. 179–185. IEEE (2010)

12. Holik, L., Lengal, O., Rogalewicz, A., Sekanina, L., Vasicek, Z., Vojnar, T.: Towards formal relaxed equivalence checking in approximate computing methodology. In: 2nd Workshop on Approximate Computing (WAPCO 2016), HiPEAC, pp. 1–6 (2016)
13. Hrbacek, R., Sekanina, L.: Towards highly optimized cartesian genetic programming: from sequential via SIMD and thread to massive parallel implementation. In: Proceedings of the 2014 Conference on Genetic and Evolutionary Computation, pp. 1015–1022. ACM(2014)
14. Kaufmann, P., Platzner, M.: Advanced techniques for the creation and propagation of modules in cartesian genetic programming. In: Genetic and Evolutionary Computation (GECCO), pp. 1219–1226. ACM Press (2008)
15. Kaufmann, P., Plessl, C., Platzner, M.: EvoCaches: application-specific adaptation of cache mappings. In: Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems, pp. 11–18. IEEE Computer Society, Los Alamitos, CA, USA (2009)
16. Kaufmann, P., Knieper, T., Platzner, M.: A novel hybrid evolutionary strategy and its periodization with multi-objective genetic optimizers. In: 2010 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8. IEEE (2010)
17. Kaufmann, P., Glette, K., Gruber, T., Platzner, M., Torresen, J., Sick, B.: Classification of electromyographic signals: comparing evolvable hardware to conventional classifiers. IEEE Trans. Evol. Comput. **17**(1), 46–63 (2013)
18. Khan, G.M., Miller, J.F., Halliday, D.M.: Evolution of Cartesian genetic programs for development of learning neural architecture. Evol. Comput. **19**(3), 469–523 (2011)
19. Knieper, T., Kaufmann, P., Glette, K., Platzner, M., Torresen, J.: Coping with resource fluctuations: the run-time reconfigurable functional unit row classifier architecture. In: Proceedings of the 9th International Conference on Evolvable Systems: From Biology to Hardware, LNCS, vol. 6274, pp. 250–261. Springer (2010)
20. Koza, J.R.: Genetic Programming: On The Programming of Computers by Means of Natural Selection. MIT press (1992)
21. Louis, S., Rawlins, G.J.E.: Designer genetic algorithms: genetic algorithms in structure design. In: Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 53–60. Morgan Kauffman (1991)
22. Markov, I.: Limits on fundamental limits to computation. Nature **512**, 147–154 (2014)
23. Miller, J.F.: An empirical study of the efficiency of learning Boolean functions using a cartesian genetic programming approach. In: Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation, vol. 2, pp. 1135–1142. Morgan Kaufmann Publishers Inc. (1999)
24. Miller, J.F.: On the filtering properties of evolved gate arrays. In: 1st NASA-DoD Workshop on Evolvable Hardware, pp. 2–11. IEEE Computer Society (1999)
25. Miller, J.F.: Cartesian Genetic Programming. Springer (2011)
26. Miller, J.F., Downing, K.: Evolution in materio: looking beyond the silicon box. In: Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware (EH'02), pp. 167–176. IEEE Computer Society (2002)
27. Miller, J.F., Smith, S.L.: Redundancy and computational efficiency in cartesian genetic programming. IEEE Trans. Evol. Comput. **10**(2), 167–174 (2006)
28. 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)
29. Miller, J.F., Thomson, P., Fogarty, T.: Designing electronic circuits using evolutionary algorithms. Arithmetic circuits: A case study. In: Genetic algorithms and evolution strategy in engineering and computer science. Wiley (1998)
30. Miller, J.F., Job, D., Vassilev, V.K.: Principles in the evolutionary design of digital circuits—part II. Genet. Program. Evolvable Mach. **1**(3), 259–288 (2000)
31. Mittal, S.: A survey of techniques for approximate computing. ACM Comput. Surv. **48**(4), 62:1–62:33 (2016)
32. Mohid, M., Miller, J.F., Harding, S.L., Tufte, G., Massey, M.K., Petty, M.C.: Evolution-in-materio: solving computational problems using carbon nanotube-polymer composites. Soft Comput. **20**(8), 3007–3022 (2016)

33. Mrazek, V., Vasicek, Z.: Automatic design of low-power vlsi circuits: Accurate and approximate multipliers. In: Proceedings of 13th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, pp. 106–113. IEEE (2015)
34. Mrazek, V., Vasicek, Z., Sekanina, L.: Evolutionary approximation of software for embedded systems: median function. In: GECCO Companion '15 Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference, pp. 795–801. ACM (2015)
35. Mrazek, V., Sarwar, S.S., Sekanina, L., Vasicek, Z., Roy, K.: Design of power-efficient approximate multipliers for approximate artificial neural networks. In: 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). pp. 811–817 (2016)
36. Mrazek, V., Hrbacek, R., Vasicek, Z., Sekanina, L.: Evoapprox8b: library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In: 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 258–261 (2017)
37. Murakawa, M., Yoshizawa, S., Kajitani, I., Furuya, T., Iwata, M., Higuchi, T.: Evolvable hardware at function level. In: Parallel Problem Solving from Nature—PPSN IV, LNCS, vol. 1141, pp. 62–71. Springer (1996)
38. Nawab, S., Oppenheim, A., Chandrakasan, A., Winograd, J., Ludwig, J.: Approximate signal processing. J. VLSI Signal Process. **15**(1–2), 177–200 (1997)
39. Nepal, K., Li, Y., Bahar, R.I., Reda, S.: ABACUS: a technique for automated behavioral synthesis of approximate computing circuits. In: Proceedings of the Conference on Design, Automation and Test in Europe, EDA Consortium, DATE'14, pp. 1–6 (2014)
40. Salvador, R., Otero, A., Mora, J., la De, E.T., Riesgo, T., Sekanina, L.: Self-reconfigurable evolvable hardware system for adaptive image processing. IEEE Trans. Comput. **62**(8), 1481–1493 (2013)
41. Sampson, A., Dietl, W., Fortuna, E., Gnanapragasam, D., Ceze, L., Grossman, D.: EnerJ: approximate data types for safe and general low-power computation. In: Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 164–174. ACM (2011)
42. Sekanina, L.: Evolvable components: from theory to hardware implementations. Nat. Comput. Ser. (2004)
43. Sekanina, L.: Evolvable hardware. In: Handbook of Natural Computing, pp. 1657–1705. Springer (2012)
44. Sekanina, L., Ruzicka, R., Gajda, Z.: Polymorphic fir filters with backup mode enabling power savings. In: Proceedings of the 2009 NASA/ESA Conference on Adaptive Hardware and Systems, pp. 43–50. IEEE Computer Society (2009)
45. Shubham, J., Venkataramani, S., Raghunathan, A.: Approximation through logic isolation for the design of quality configurable circuits. In: Proceedings of the 2016 Design, Automation & Test in Europe Conference and Exhibition (DATE), pp. 1–6. EDA Consortium (2016)
46. Sikulova, M., Sekanina, L.: Acceleration of evolutionary image filter design using coevolution in Cartesian GP. In: Parallel Problem Solving from Nature-PPSN XII, no. 7491 in LNCS, pp. 163–172. Springer (2012)
47. Slany, K., Sekanina, L.: Fitness landscape analysis and image filter evolution using functional-level CGP. In: Proceedings of European Conference on Genetic Programming. LNCS, vol. 4445, pp. 311–320. Springer (2007)
48. Soeken, M., Grosse, D., Chandrasekharan, A., Drechsler, R.: BDD minimization for approximate computing. In: 21st Asia and South Pacific Design Automation Conference ASP-DAC 2016, pp. 474–479. IEEE (2016)
49. Stomeo, E., Kalganova, T., Lambert, C.: Generalized disjunction decomposition for evolvable hardware. IEEE Trans. Syst. Man Cybern. Part B **36**(5), 1024–1043 (2006)
50. Thompson, A., Layzell, P., Zebulum, S.: Explorations in design space: unconventional electronics design through artificial evolution. IEEE Trans. Evol. Comput. **3**(3), 167–196 (1999)
51. Turner, A.J., Miller, J.F.: Recurrent cartesian genetic programming. In: Parallel Problem Solving from Nature—PPSN XIII, pp. 476–486. Springer (2014)

52. Turner, A.J., Miller, J.F.: Neutral genetic drift: an investigation using cartesian genetic programming. Genet. Program. Evolvable Mach. **16**(4), 531–558 (2015)
53. 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)
54. Vasicek, Z., Sekanina, L.: An evolvable hardware system in Xilinx Virtex II Pro FPGA. Int. J. Innov. Comput. Appl. **1**(1), 63–73 (2007)
55. Vasicek, Z., Sekanina, L.: Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. Genet. Program. Evolvable Mach. **12**(3), 305–327 (2011)
56. Vasicek, Z., Sekanina, L.: A global postsynthesis optimization method for combinational circuits. In: Proceedings of the Design, Automation and Test in Europe, DATE, pp. 1525–1528. IEEE Computer Society (2011)
57. Vasicek, Z., Sekanina, L.: Evolutionary design of approximate multipliers under different error metrics. In: IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems 2013, pp. 135–140. IEEE (2014)
58. Vasicek, Z., Sekanina, L.: How to evolve complex combinational circuits from scratch? In: 2014 IEEE International Conference on Evolvable Systems Proceedings, pp. 133–140. IEEE (2014)
59. Vasicek, Z., Sekanina, L.: Circuit approximation using single- and multi-objective cartesian GP. In: Genetic Programming. LNCS 9025, pp. 217–229. Springer (2015)
60. Vasicek, Z., Sekanina, L.: Evolutionary approach to approximate digital circuits design. IEEE Trans. Evol. Comput. **19**(3), 432–444 (2015)
61. Vasicek, Z., Sekanina, L.: Evolutionary design of complex approximate combinational circuits. Genet. Program. Evolvable Mach. **17**(2), 169–192 (2016)
62. 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)
63. Vassilev, V., Job, D., Miller, J.F.: Towards the automatic design of more efficient digital circuits. In: Proceedings of the 2nd NASA/DoD Workshop on Evolvable Hardware, pp. 151–160. IEEE Computer Society (2000)
64. Vazirani, V.V.: Approximation Algorithms. Springer (2001)
65. Venkataramani, S., Roy, K., Raghunathan, A.: Substitute-and-simplify: a unified design paradigm for approximate and quality configurable circuits. Design, Automation and Test in Europe, DATE'13, pp. 1367–1372. EDA Consortium San Jose, CA, USA (2013)
66. Venkatesan, R., Agarwal, A., Roy, K., Raghunathan, A.: MACACO: modeling and analysis of circuits for approximate computing. In: 2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 667–673. IEEE (2011)
67. Walker, J.A., Miller, J.F.: The automatic acquisition, evolution and reuse of modules in cartesian genetic programming. IEEE Trans. Evol. Comput. **12**(4), 397–417 (2008)
68. Walker, J.A., Hilder, J.A., Tyrrell, A.M.: Evolving Variability-Tolerant CMOS Designs. Springer, Berlin Heidelberg (2008)
69. Walker, J.A., Hilder, J.A., Reid, D., Asenov, A., Roy, S., Millar, C., Tyrrell, A.M.: The evolution of standard cell libraries for future technology nodes. Genet. Program. Evolvable Mach. **12**(3), 235–256 (2011)