Genetic Programming and Evolvable Machines (2022) 23:453–493
https://doi.org/10.1007/s10710-022-09436-w

A novel tree-based representation for evolving analog circuits and its application to memristor-based pulse generation circuit

Xinming Shi[1,2] · Leandro L. Minku[2] · Xin Yao[1,2]

Received: 15 August 2021 / Revised: 16 January 2022 / Accepted: 24 April 2022 /
Published online: 21 July 2022

Abstract
When applying evolutionary algorithms to circuit design automation, circuit representation is the first consideration. There have been several studies applying different circuit representations. However, they still have some problems, such as lack of design ability, which means the diversity of evolved circuits was limited by the circuit representation, and inefficient transformation from circuit representation into SPICE (Simulation Program with Integrated Circuit Emphasis) netlist. In this paper, a novel tree-based circuit representation for analog circuits is proposed, which is equipped with an intuitive and three-terminal devices friendly mapping rule between circuit representation and SPICE netlist, as well as a suitable crossover operator. Based on the proposed representation, a framework for automated analog circuit design using genetic programming is proposed to evolve both the circuit topology and device values. Three benchmark circuits are applied to evaluate the proposed approach, showing that the proposed method is feasible and evolves analog circuits with better fitness and number of components while using less fitness evaluations than existing approaches. Furthermore, considering physical scalability limits of conventional circuit elements and the increased interest in emerging technologies, a memristor-based pulse generation circuit is also evolved based on the proposed method. The feasibility of the evolved circuits is verified by circuit simulation successfully. The experiment results show that the evolved memristive circuit is more compact and has better energy efficiency compared with existing manually-designed circuits.

Keywords Automated analog circuit design · Memristor · Circuit representation · Genetic programming · Evolvable hardware

Xin Yao
xiny@sustech.edu.cn

Extended author information available on the last page of the article

1 Introduction

Circuit design automation has attracted increasing attention, with analogue circuit design being particularly challenging due to its complex topology and parameter selection [1]. There have been several methodologies to study analog circuit design automation, including methods that incorporate domain knowledge [2], evolutionary algorithms [3] and simulated annealing [4]. Some of them are based on domain knowledge and require significant expertise to be applied. For example, Oliver et al. [2] proposed a constraint-driven method to implement the automatic design of analog circuits. However, this type of method requires abundant knowledge of circuit design to be developed and limits the range of exploring circuits.

Evolutionary algorithms have been widely applied to automated analog circuit design, requiring neither design rules nor domain knowledge from experts [5]. For example, Kruiskamp et al. [6] proposed the prototype synthesis tool DARWIN based on Genetic Algorithm (GA) to design the CMOS opamp. Grammatical Evolution (GE) [7] and Genetic Programming (GP) [8] have also been used successfully for the automated design of analog circuits. Some researchers commented that GP makes it possible for generating topology of analog circuits with arbitrary connections [9]. Several earlier studies proposed that GP is likely the most successful evolutionary computation-based paradigm for analog circuit synthesis [10], given its good diversity for automated analog circuit design [11, 12]. Even in recent years, GP (and its variants) is still considered as the successful evolutionary paradigm for analog circuit design [13].

Circuit representation is the first consideration in automated circuit design. It indicates how to encode a circuit. According to different data structures, there are several types of circuit representations, such as string-based [10, 14], tree-based [12, 15, 16] and graph-based circuit representations [17, 18]. However, there are also some limitations of these different types of circuit representations. Gan et al. [17] suggested that a string-based representation [14] is so complex that much computation time is taken during the decoding process. The linear string-based representation has another limitation: it cannot support all possible circuit topologies [10]. In addition, some researchers [9, 16, 17] proposed that the tree-based circuit representations also have some inadequacies, such as bloat size of the evolved design [15], limited application [12], inefficient crossover operators and complex transformation into circuit netlists [11]. As for graph-based circuit representation, designing good crossover operators is a challenging problem, as it may result in infeasible individuals during the evolutionary processes [18].

Therefore, a new circuit representation for automated analog circuit design based on GP is proposed in this paper. The proposed representation has the following characteristics. First, it makes the transformation between the circuit representation and the circuit netlists more direct and efficient, which is a desirable property for the evolution of circuits. Second, the proposed circuit representation can be applied in the circuits with either two-terminal or three-terminal electrical elements. Finally, the novel tree-based circuit representation has more suitable crossover operator compared with some existing representations, potentially aiding the evolutionary process to obtain better circuits.

Based on the proposed circuit representation, a suitable evolutionary approach based on GP is designed, which includes topology evolution and device value optimization. We validate the proposed representation and its corresponding GP based on three widely used benchmark circuits [13, 14, 19], showing that the evolved results based on proposed method are better in terms of the number of evaluations and circuit performance than those obtained by existing methods [13, 14, 19]. The benefits of the characteristics of our proposed circuit representation are also verified by comparison experiments.

In addition, we perform a case study of using the proposed approach to evolve memristor-based circuits, which are particularly relevant nowadays given the rapid development of emerging electronic devices, the wide application prospects of memristors [20] and the increasing trend of researching neuromrophic computing [21–23]. Considering the dynamic characteristics of memristors, there are two parts of challenges of designing memristor-based circuit manually, which are how to tune into the specific memristance by control circuits or signals, and how to manipulate the state switching behaviors of different memristors (from HRS/LRS to LRS/HRS), respectively. As for the manual design, it is intensive and time-consuming for designers to manipulate these different characteristics of memristors during the designing process. And some devices with bulk size, such as capacitors, even external fabricated chips, are always selected to construct circuits [22, 24], incurring large size and high power consumption of designed circuit system. Therefore, it is reasonable to apply the evolutionary approach to manipulate different characteristics of memristors and other devices to design memristor-based circuits pursuing the functionality, energy efficiency, and compactness synchronously.

There are very few studies on automated design of memristor-based analog circuits [25, 26]. Some of the existing work [25] is based on Koza's tree representation, which has the above mentioned limitations. Moreover, most existing work focuses on synthesising memristive digital circuits [27, 28], particularly logic circuits. There are very few works attempting to automatically design analog memristive circuits as done in our paper. Our study shows that our proposed method can successfully be applied to evolve emerging device-based analog circuits, obtaining better results compared with the manual-design circuits for the similar purpose in terms of the number of components applied and power consumption.

Overall, our key contributions are threefold:

- A novel tree-based representation that can be transformed more directly and efficiently to circuit netlists, and for which a suitable crossover operator can be applied, better supporting automated circuit evolution.
- A GP framework that uses the proposed representation and can improve the number of evaluations, fitness and hits over existing literature.
- A case study applying the proposed GP framework to analog memristive circuit design, demonstrating that the proposed framework can lead to more compact size and greater energy efficiency than existing manually designed circuits.

Table 1 Characteristics of automated circuit design methodologies based on domain knowledge and EA [29]

Based methods	Structures	Complexity	Knowledge required
Knowledge	Familiar	Very high	Very high
EA	Unfamiliar	Low	Low

The rest of the paper is structured as follows. Section 2 presents related work. Section 3 introduces our proposed circuit representation in detail. Section 4 explains the evolutionary design framework of analog circuits based on GP proposed in this work. Section 5 focuses on the validation of our method, where three benchmark circuits are applied and the comparisons with previous works in the field are presented. Section 6 shows how the proposed method is applied to generating a memristor-based pulse generation circuit, and presents comparisons with manual-design circuits. The conclusion of our work is presented in Sect. 7.

2 Related work

2.1 Automated analog circuit design algorithms

The complex topology and a large number of component values for circuits makes its automated analog design a challenge. Considering different strategies to implement automated analog circuit design, researchers specified several ways to tackle the challenge, including domain knowledge-based and evolutionary algorithm (EA)-based approaches [29]. Table 1 gives a brief comparison of automated circuit design methodologies based on domain knowledge and EA.

As the name implies, knowledge-based methods rely on the knowledge obtained from specific circuits or sub-circuits, thus the great human effort and experience is highly required to extract the knowledge for each generated structure one by one [29], which is unfriendly to inexperienced developers. In addition, the computing complexity of some knowledge-based methods [30] is high, as it needs to check if the automatically generated part is potentially useful by circuit knowledge such as transmission parameters, incurring high computing complexity.

Compared with knowledge-based methods, EA-based approaches are more independent of human effort [29]. EA-based methods require less human effort, and the structure of evolved circuit is unfamiliar compared with that of knowledge-based methods. Unfamiliar structures of circuit can enrich the candidates of desired circuits, which may find better results among them than that of familiar design. Based on various evolutionary algorithms, different works of automated circuit design have been proposed [6, 8, 13, 31]. In Kruiskamp et al. [6], applied Genetic Algorithms (GA) to solve CMOS opamp synthesis problem, where each individual in the population has a multi-gene chromosome that can be converted into a corresponding circuit by decoding it. Grimbleby [32] used GA for automated analogue network synthesis, where GA was used to configure the circuit structure. However, component values should be determined by subsequent numerical optimization.

Genetic programming (GP) has been widely used in automated circuit design and other fields like signal processing and system structure identification [33]. Different from numerical optimization algorithms, GP can design complex structures starting with programs or operations as genes and evolve to better programs by using GA-liked operators like selection, mutation, and crossover. In [11], Koza used GP to design eight different types of circuits automatically with minimal problem-specific information, demonstrating the general applicability of GP to solve the problem of automated synthesis of analog electrical circuits. In recent years, GP (and its variants) is still considered as the evolutionary paradigm with more successful results in the field of analog circuit design [13].

The differences between GA and GP in terms of evolving analog circuit mainly reflect on the representation. Specifically, GA employs string-based representations of binary or float value to the evolution [34, 35]. String is an efficient structure of value optimization, therefore, some GA studies only provide the solutions to the devices value with the already known circuit topology [6, 32]. Different from these GA studies, GP approaches can generate the circuit netlists for evolution, which contain both of the circuit device value and circuit topology. Using GA, some researchers encode the sequence of the string as the circuit topology and its string value encoded as the device parameter, which incurs the limited circuit diversity [10] and complex procedure of decoding to its phenotype [17]. Different from these GA studies, in GP studies, the unbalanced tree structure of function nodes encodes the relative positions of devices and the terminal nodes further encode the specific positions for the circuit netlists. Therefore, it allows greater circuit diversity and more efficient decoding procedure to phenotype.

2.2 Circuit representation

Circuit representation is indispensable for automated analog circuit design, as it allows an evolutionary algorithm to alter the circuit topology and parameters by making use of genetic operators. According to different data structures, there are several types of circuit representation:

1. String-based circuit representation: Mattiussi et al. proposed the Analog Genetic Encoding (AGE) approach based on string representation to synthesize analog circuits [14]. Each gene of AGE's genome denotes a device, in which two regions are included for denoting each terminal and parameters of the device. Connection of these devices is determined by a device interaction map. Final circuit is constructed by connecting all devices step-by-step according to the device interaction map. The experimental results show that AGE can synthesize analog electrical circuits. However, this representation is so complex that much computation time is taken during decoding process [17].

   The linear representation proposed by Jason in 1999 [10] is also based on string structure. It can encode the circuit construction operations into opcodes, such as *x-move-to-new*, *x-cast-to-previous* and *x-cast-to-ground*, *x-cast-to-input* and *x-cast-to-output*. Here, *x-move-to-new* represents adding a device x to the active node and generating a newly active node; *x-cast-to-previous* represents inserting a device *x* between the active node and the circuit under construction; *x-cast-to-ground* represents casting a device *x* between the active node and ground; *x-cast-to-input* represents casting a device *x* between the active node and input; and *x-cast-to-output* represents casting a device *x* between the active node and output. However, this representation cannot support all possible circuit topologies, being of limited applicability [10]. Specifically, the circuit construction are guided by above-mentioned five basic instructions, and the circuit evolution only happened between the active node of constructed part and new devices, by which some of the connections cannot be established.

2. Tree-based circuit representation: Koza and his collaborators [8] have done several studies on automated synthesis of analog circuits by means of tree-based GP. Sripramong et al. [36] improved on Koza's work by using the same tree-based circuit representation, but with a recursive analysis to verify circuit correctness. In [37], individuals were also represented by Koza's tree-based representation, where non-leaf nodes represent circuit components or connection ways, and leaf nodes are defined as terminal nodes or constants. However, Koza's tree-based representation is prone to a phenomenon named bloat, which refers to circuits often growing excessively large [15], which may lead to high cost in terms of larger circuit area and power consumption.

   In 2006, Chang et al. [12] proposed a novel tree-based representation for synthesizing RLC circuit. However, this representation has been criticized for lack of design ability [16] and can only be applied in the two-pole electrical elements [9]. As the length of the tree is dynamic, the potential of bloat still exists.
3. Graph-based circuit representation: Graphs are attractive representations for circuits as they are compact and intuitive for representing circuits [38]. An Evolutionary Graph Generation (EGG) system was proposed to synthesise digital circuits [39, 40]. This EGG system was also applied to synthesise analog circuits [41]. In EGG, nodes represent device or I/O pin, edges represent connection between devices and I/O pins, thus individuals are represented as graph. However, the validity of individuals cannot be guaranteed during evolution, because the main genetic operators such as crossover and mutation may break the rationality behind the individual circuit [17].

   Gan et al. [17] used graph representation to automatically synthesize passive analog filters. The circuit topologies can be modified by six types of mutation operations. Combined with clone selection algorithm, components parameters could be synthesized simultaneously [17]. However, only the mutation operations were employed to modify the individuals. The operation of crossover was ignored in this work, which may limit to obtain better results. Some researchers also proposed that the node incidence matrix relates the vertices to the edges of a graph, possessing structural properties that are not, generally, preserved after subjected to a crossover operation. This will result in invalid individuals in evolutionary processes [18]. And as for some graph representation based on adjacency matrix, they are unable to represent three terminal elements [17].

   As for graph structure to represent analog circuits, there might be a number of no connections among the circuit devices, which potentially incurs waste of storage resources. Moreover, the graphs are not hierarchical structures. Therefore, it is difficult to use them as structures that support modularity, where we can easily identify different sub-components of the circuit. This means that it would be difficult to design a crossover operator that does not break the circuit based on graph representation. The trees, on the other hand, are hierarchical structures. Each sub-tree can represent a different sub-component of the graph. This kind of structure lends itself to crossover operators that would not produce infeasible circuits, such as the ones proposed in this paper.

Graph-based CGP has also been applied for evolving analog circuits [42], where they have applied a modified circuit representation. This modified circuit representation has their limitations. First, the circuit device values are evolved based on a look-up table, which is discrete. Moreover, decoding procedures contain 3 stages, being complex. And also decoding constraints are designed for Mosfets, not being compatible with two-terminal devices like resistor and memristor.

In addition, dealing with infeasible circuits is also an important part of circuit representation. The feasible circuit individuals should follow some rules based on circuit theory, such as avoiding short-circuit and open-circuit. There are several approaches of dealing with circuit feasibility, such as discarding infeasible circuit individuals [9], accepting infeasible circuit individuals with penalty [17], and repairing infeasible circuit individuals [10, 14, 43]. Rojec et al. [9] applied the discarding strategy, where only the individuals that pass the connection detection can be evaluated, the other individuals left in the population are discarded. However, this may limit the diversity of the initial population. Gan et al. [17] applied the penalty strategy, where the individuals that have no connection with at least one of the accessible nodes are accepted but penalized. Several studies applied repairing strategies. There are three types strategies of repairing strategies: (1) deleting the dangling terminals in the infeasible circuits [43, 44]; (2) connecting the dangling terminals to the existing circuit nodes [10]; and (3) , inserting the dangling terminals into a resistor with large value to make the circuit simulation valid [14].

2.3 Automated memristor-based circuit design

Memristor is the fourth basic element in electronic circuit besides resistor, capacitor and inductor. The ability of memristors to act as thresholded electrically tuneable, multilevel, non-volatile resistive loads, combined with their inherently scaling-friendly and low power [45, 46] has rendered them a highly promising candidate for use in future electronics applications. There are two main problems of designing memristor-based circuits, which are how to tune into the specific analog memristance by control circuits or signals, and how to manipulate the state switching behaviors of memristors (from HRS/LRS to LRS/HRS), respectively. As for the former problem, some researchers have designed the memristance tuning analog circuits to control the memristance variation [47]. As for the latter problem, researchers have applied the state switching behaviors with different rates to implement different functions like neuronal exponential spike generation [23] and logic circuits [24].

At present, most of memristor-based circuits are designed manually. However, manual design of memristor-based circuits is a time-consuming task, because the target functions, size and power consumption need to be taken into consideration at the same time during the design process. Moreover, designing multi-memristor combinations with different connections is also a difficult problem for its complex dynamic electronic characteristics. Therefore, it is valuable to apply evolutionary approaches to automatically generate memristor-based circuits. However, there are few studies related to the memristor-based automated circuit design. Some researchers have applied evolutionary approaches to synthesize memristor-based digital circuits, particularly logic circuits [27, 28]. There are two works focusing on analog circuits [25, 26]. They are based on the traditional Koza's tree representation, which suffer from the problems of that representation mentioned in Sect. 2.2.

3 A novel tree-based circuit representation

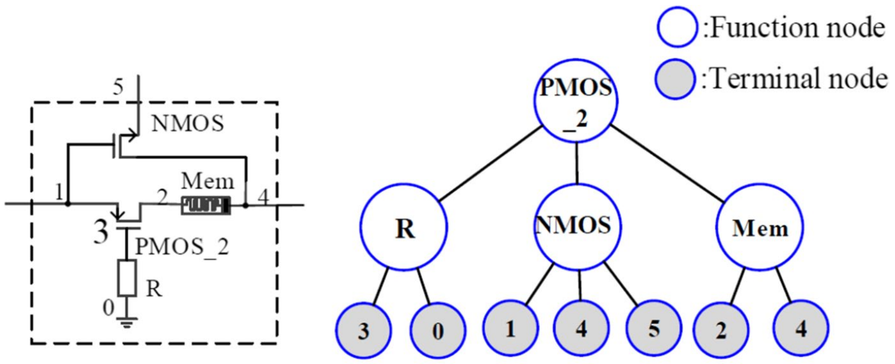
Ideally, both of the circuit topology evolution and device value optimization should be considered in analog circuit design automation, avoiding the need for expensive human design knowledge. In this section, we propose a novel tree-based circuit representation for evolving analog circuits that can be used to evolve both the circuit topology and device values. The details of representing analog circuits (Sect. 3.1), structural plausibility check to make sure each generated circuit tree is corresponding to a valid circuit (Sect. 3.2), and the advantages of the proposed tree-based representation (Sect. 3.3) will be introduced.

3.1 Circuit representation and netlist transformation

The circuit representation is based on a multi-tree. We define the non-leaf node as function node and leaf node as terminal node (see right part of Fig. 1).

A function node (Fig. 2a) consists of two parts, the first part is a device type depending on the circuits to be implemented, for example, in a simple RC filter circuit, the device type of a function node will be R (resistor) or C (capacitor). The second part is the value tree representing the value of the circuit device in the form of a binary tree, as shown in the example provided in Fig. 3. Every value-based device will have its corresponding value tree. For devices that do not require a value, the value tree is null.

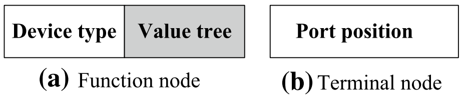
A single terminal node (Fig. 2b) refers to the position of one device port in the circuit netlist, which is represented by an integer number. According to the number of device ports, each function node has a defined arity (i.e., number of child nodes). For example, memristor, a two-port device, can be represented by an arity-2 node in the circuit representation. Similarly, MOSFET, a three-port device, can be represented by an arity-3 node in the circuit representation. Therefore, the circuit connection of a device in the circuit netlist can be determined by assigning the position numbers of each port (Fig. 4).



Device	Terminals		
	Terminal1	Terminal2	Terminal3
PMOS_2	3	1	2
R	3	0	-
NMOS	1	4	5
Mem	2	4	-

Fig. 1 A circuit example represented by proposed method

Fig. 2 Diagram of function node and terminal node



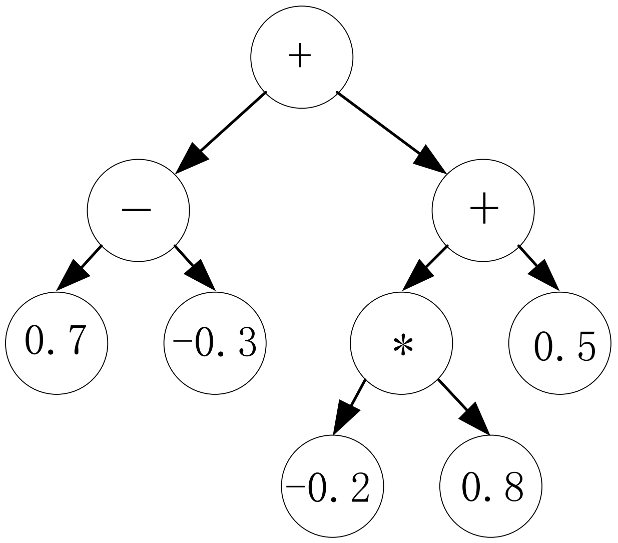
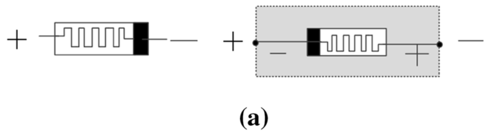


Fig. 3 An example of value tree



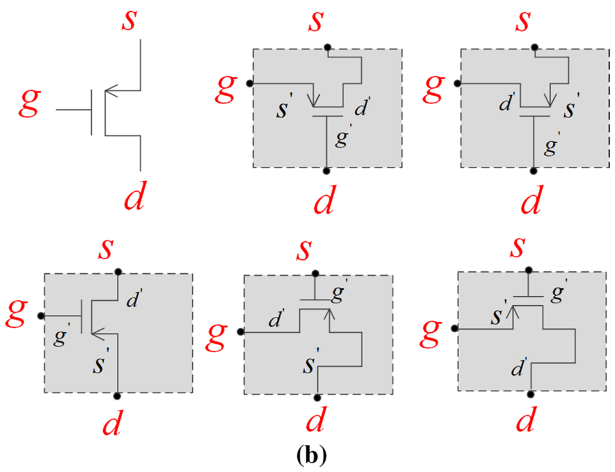


Fig. 4 Diagram of equivalent devices with different terminal definitions. **a** Memristor and its equivalent device with reversed polarity. **b** PMOS and its five types of equivalent device

In order to enhance the flexibility of circuit representation, the devices with polarity correspond to different function nodes. For instance, the memristor and its equivalent memristor with reversed polarity in Fig. 5a are represented by two different function nodes. Similarly, the PMOS with original definition and its five types of equivalent PMOS with different port definitions are also represented by fix different function nodes. In this way, the circuits with any-connection can be implemented.

During the process of converting the tree-based circuit representation into the circuit netlists, each function node needs to be specified a netlist position number. Therefore, a hierarchical relationship between parent node and child node is defined. The terminals of the parent node are formed by the left terminals of its child nodes, which allows for the representation to be circuit-intuitive. Figure 1 shows an example of how to convert the devices in a tree-based circuit representation into their corresponding circuit position numbers. In Fig. 1, $PMOS_2$ is the parent node of *R*, *NMOS*, and *Mem*. According to the hierarchical relationship between parent node and child node, *Terminal1* of $PMOS_2$ will be *3*, which is the same as *Terminal1* of *R*; *Terminal2* of $PMOS_2$ will be *1*, which is the same as *Terminal1* of *NMOS*; and *Terminal3* of $PMOS_2$ will be *2*, which is the same as *Terminal1* of *Mem*. In this way, all the function nodes will be assigned corresponding terminal nodes, enabling the construction of a circuit netlist further.

In Koza's tree-based structure [8], both of the circuit constructing operations and the circuit devices are function nodes. However, the circuit constructing operations such as series, parallel and flip will use many nodes within a tree, which may cause some

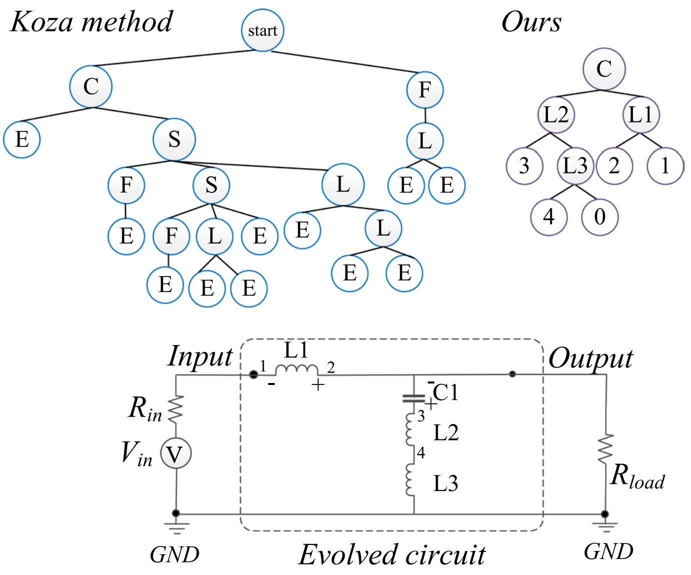


Fig. 5 An example of encoding a same circuit by Koza's and our method

undesirable scenarios where a complex tree with plenty of circuit constructing operations and fewer circuit components can only represent a simple circuit. These undesirable scenarios can be prevented in our representation, by using different function nodes of a tree to represent circuit components and use the topology of a tree to represent the circuit connection. Therefore, we can use a simpler tree to represent the same circuit compared with Koza's tree. Figure 5 gives an example of encoding a given circuit by Koza's and our method. We can see that many circuit construction operation nodes are used by Koza's method to construct the circuit, such as S (series), F (flip) and E (end). As for our proposed method, only the circuit devices and their netlist positions are necessary to construct the circuit, which is therefore more compact than Koza's tree.

Besides the above tree structure representing the circuit topology, the value tree, which is embedded in the related function nodes, is designed to represent the device value. In the value tree, the arithmetic operators are regarded as function nodes, and the terminal nodes are float numbers from − 1 to 1. The equation to calculate the value of a tree is:

$$Value_{device} = A \times \left(10^{Value_{tree}}\right), \tag{1}$$

where A is the fitting parameter that is capable of scaling the obtained value into the reasonable range of corresponding devices, and $Value_{tree}$ can be calculated by the arithmetic operators and float numbers in the value tree. Taking Fig. 3 as an example and assuming $A = 10^5$, the $Value_{tree}$ of a resistor is calculated as:

$$Value_{device} = 10^5 \times (10^{((-0.2*0.8)+0.5)+(0.7-(-0.3))}). \tag{2}$$

The value tree allows the approach to optimise by shuffling around arithmetic operations from a small and tractable set, as opposed to randomly having to “mutate" values directly. This can protect those “good" arithmetic operations that are beneficial for the evolution, generating better value gradually during the evolution.

3.2 Three types of structure check

We want to prevent the bloat problem [48] of tree structure and the invalid circuits that will cause simulation failure, therefore, three types of structure check are applied in this work. In order to check if a circuit is feasible, the usage count of different terminal nodes is an important indicator. Algorithm 1 gives how to count the terminal nodes of a tree, where the post-fix traversing is applied to each parent node. When the current traversed parent node is the terminal node, the current parent node will be added to the terminal set (Ln 2–3 in Algorithm 1). When the current traversed parent node is the function node, every child nodes of this parent node will be traversed. The value embedded on the terminal node will be stored in *portList*. The counts of each element in *portList* will be updated to *terminal_count* (Ln 8–9 in Algorithm 1). Three types of structure check are introduced accordingly.

First, our representation considers that there is a predefined embryo circuit representing the initial circuit configuration of the to-be evolved circuit. The embryo circuit design is generally simple, which could be specified depending on different circuit requirements [11, 13, 14]. The common embryo circuit includes voltage sources, ground, and load resistor. This part of circuit will not be evolved, i.e., will remain fixed during the evolutionary process. However, the way with which the to-be evolved circuit will be connected with the embryo will be determined by the evolutionary process. Therefore, the external nodes of embryo circuit are defined as necessary nodes. And we define the unnecessary nodes from two perspectives, which are node counts and node function. Specifically, if a node that has been assigned over twice or does not belong to the embryo circuit will be regarded as unnecessary node. The first step is that the external terminals of the embryo circuit (necessary nodes) should be checked if they are all assigned to the tree terminal node by the evolutionary process. If they are not, the unconnected terminal of embryo circuit will be isolated, incurring invalid circuits during the evolution. Such infeasible individuals are repaired by replacing a random unnecessary node by the isolated necessary node. The *step 1* in Fig. 6 gives an example of the first structure check. Nodes 0, 1, 2, 3 indicates the embryo nodes that must be assigned in the tree necessarily. However, as we can see that the node 1 has not existed in the tree. Therefore, the unnecessary nodes, which are 5, 7, 9, 6, 8, 15, 12, one of them will be selected randomly and replace it by node 1. In this way, all the external terminals of the embryo circuit will be connected to the evolved circuit.

Second, in our circuit representation, the terminal nodes represent the port position directly. Therefore, we need to make sure the devices are connected to each other composing a complete circuitry and preventing the problem of hang terminals, i.e., isolated terminals of devices that are not connected to the evolved circuit. In particular, for this problem not to happen, one terminal node should always be assigned at least twice in a

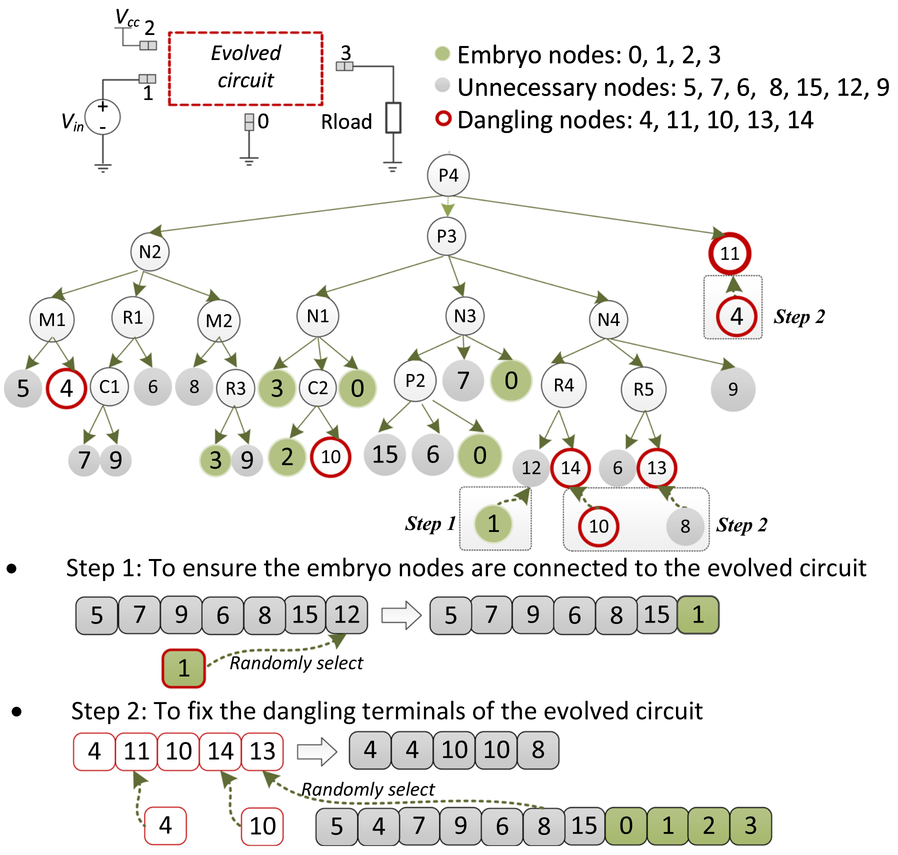


Fig. 6 Diagram of tree structure check

single tree. Therefore, we have to make sure that the random-generated terminal nodes of the tree-based circuit representation appear in the tree at least twice. The *step 2* in Fig. 6 shows an example of the second structure check works. Nodes 4, 11, 10, 13, 14 are dangling terminals without the connections with other nodes. The last node 13 will be fixed firstly, where a random node from unnecessary nodes and embryo nodes will substitute the node 13. Then, the neighboring two nodes will be set as the same. As shown in Fig. 6, 4, 11, 10, 13, 14 are changed to 4, 4, 10, 10, 8, respectively. By the tree structure check, all the embryo nodes could be connected into the evolved circuit and there will be no dangling terminals existed in the evolved circuit.

Algorithm 1 Pseudo code of nodes count function *update_terminal*()

```
def $update\_terminal(parent : Node)$:
if parent.type == ‘terminal’ then
    $terminal\_set.append(parent)$
else
    if parent.type == ‘function’ then
        for node in parent.childList do
            $update\_terminal(node, Calparent)$
            for port in node.portList do
                $terminal\_count[port] += 1$
            end for
        end for
    end if
end if
```

Third, some researchers asserted that the restriction of tree depth or the number of nodes can ameliorate the bloat problem of tree [9]. The third type of structure check is to apply a max depth limit for the tree. Therefore, the depth of a tree will be limited in the range between minimum and maximum. After generating a new tree, the tree depth will be checked and the part that exceeds the max depth will be replaced by its left terminal node.

3.3 Advantages of our proposed tree-based representation

The advantages of our proposed tree-based representation are as follows:

- Efficient transformation into circuit netlists:

  In Koza's tree representation [19], function nodes contain both of the circuit construction operations, such as parallel (P) or series (S) operations, and device types, such as resistor (R) or capacitor (C), which leads to complex tree structures. Compared with Koza's tree representation, the function nodes of our proposed tree-based representation only contain circuit devices manipulating different ways of connections between function nodes and terminal nodes (port position) to represent circuits, which leads to more compact tree structure. We only consider the two-terminal and three-terminal devices are applied in this work. The same height H for both of our proposed tree-based representation and Koza's tree are given for the analysis. When the circuit devices are all the two-terminal devices, there will be $\frac{2^H-2}{2}$ devices, and when the circuit devices are all the three-terminal devices, there will be $\frac{3^H-3}{6}$ devices. Therefore, the number of circuit elements that our proposed tree can represent will be in the range of $\left(\frac{2^H-2}{2}, \frac{3^H-3}{6}\right)$. However, as the function nodes of Koza's tree not only contain the device types but also the circuit construction operations, the number of the circuit elements that Koza's tree can represent will be in the range of $\left(\lambda \times \frac{2^H-2}{2}, \lambda \times \frac{3^H-3}{6}\right)$, where λ is a constant indicating the proportion of the number of device types to all the number of function nodes. Theoretically, λ could be 1, indicating the function nodes are all the circuit elements.

Actually when $\lambda = 1$, there will be only one device in the circuit generated by Koza's method, which is unlikely to perform the desired function of the circuit. Therefore, $\lambda \in (0, 1)$. As a results, our proposed tree-based representation is more compact, which means it can represent more devices in a circuit with the same tree height. More compact tree structures make transformation into netlists more efficient.

In addition, the form of "Device–circuit position" of our proposed representation also leads to more efficient transformation into netlists. Specifically, the executable circuit netlist could be transformed from the form of "Device–circuit position" of our proposed representation directly by the post-order traversing. However, Koza's tree-based representation [11] has tree structures based on the form of "Circuit construction operations–devices", which require extra steps to be executed in addition to traversing the whole tree in order to assign the corresponding terminal position to circuit devices for constructing or updating the circuit netlist. Therefore, considering the more compact tree structure and the direct form of "Device–circuit position", our proposed tree-based representation can make the transformation into circuit netlist more efficient.

- Support for both of two-terminal and three-terminal devices

  As mentioned in Sect. 3.1, the proposed representation can be used both with two-port (resistor or capacitor) and three-port (transistor) electrical devices. Different from topology-restricted methods [12], the proposed circuit representation can be applied in both of two-port and three-port based topologies. Therefore, the proposed method has a wider application range of circuit design.
- Suitable and efficient crossover operators for circuit evolution

  Some researchers proposed that it is challenging to design crossover operators for graph-based circuit representation due to its close-loop structure [38]. Thus, only the point mutation operations are applied in some work [17]. Compared with graph-based representations, the proposed circuit representation lends itself better to crossover operators, which may help the evolutionary process to find better circuits. The sub-circuits of a circuit can be represented by the branches of the tree directly in our proposed representation, facilitating the design of suitable crossover operators for circuit evolution. However, in Koza's tree representation [11], the sub-circuits of the whole circuit are represented not only by their corresponding branches but also by the function nodes in the parent level of the branches, which may lead to inefficient crossover operations. The crossover operator adopted with our representation is explained in Sect. 4.3.

4 Evolutionary design of analog circuits based on GP

In this section, we will introduce a GP-based circuit evolutionary algorithm based on the proposed circuit representation. The schematic flow of the algorithm is given in Fig. 7, which explains how the circuit topology (Sect. 4.3) and device value (Sect. 4.4) are evolved.

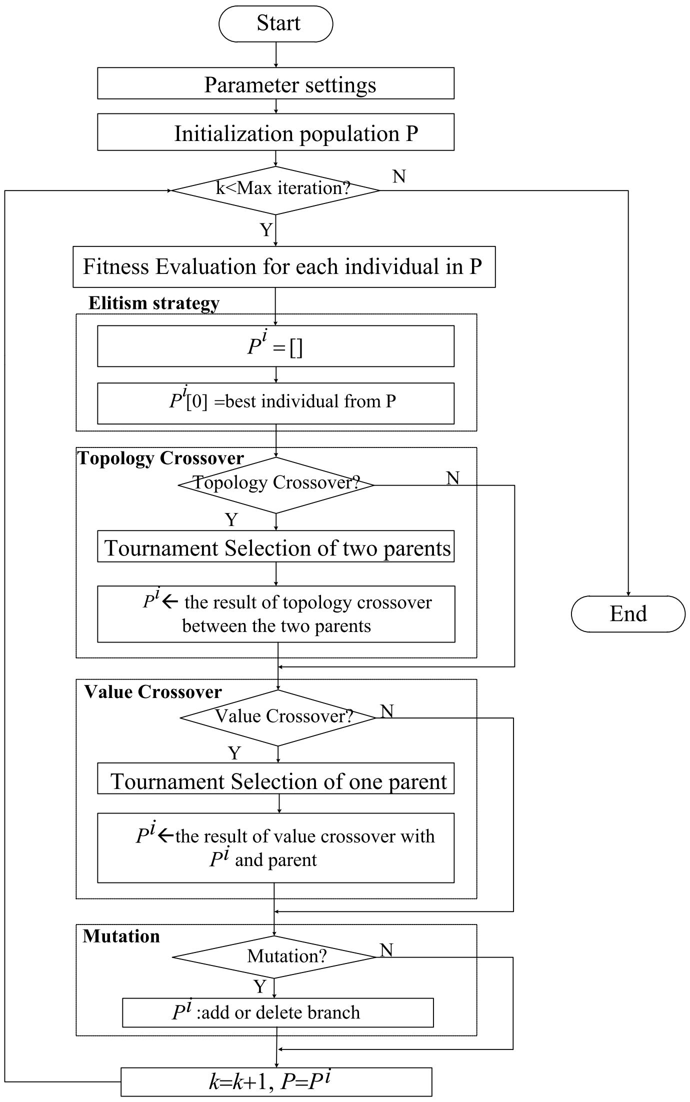

Start
Parameter settings
Initialization population P
k<Max iteration?
N
Y
Fitness Evaluation for each individual in P
Elitism strategy
$P^i = []$
$P^i[0]$ =best individual from P
Topology Crossover
Topology Crossover?
N
Y
Tournament Selection of two parents
$P^i \leftarrow$ the result of topology crossover between the two parents
End
Value Crossover
Value Crossover?
N
Y
Tournament Selection of one parent
$P^i \leftarrow$the result of value crossover with P^i and parent
Mutation
Mutation?
N
Y
P^i :add or delete branch
$k=k+1$, $P=P^i$

Fig. 7 Flowchart of GP algorithm for evolving analog circuits

4.1 Overall flowchart

As shown in Fig. 7, a GP algorithm for evolving analog circuits contains the following steps:

Step 1: Parameter settings—a series of parameters is set, such as population size N, tournament size T, max iterations M, max depth of tree H, topology crossover rate P_{cross}, mutation rate P_{mutate} and value crossover rate $P_{valuecross}$.

Step 2: Population initialization—According to the parameters set in Step 1, the population is initialized. The details of population initialization in our proposed GP design are introduced in Sect. 4.2.

Step 3: Fitness evaluation—Each individual is assigned a fitness value by the proposed fitness function. The specified fitness functions for different tasks are proposed in Sect. 5.

Step 4: Elitism strategy—The fourth step is to employ the elitism strategy. All the individuals in the population are sorted by their fitness value and the one with the best fitness is reserved.

Step 5: Topology crossover—The next step is topology crossover. Two parents are selected by n-tournament selection to go through topology crossover with probability P_{cross}. One child is generated as a result of the crossover operation, and will survive to the next generation. Moreover, the generated child will replace the parent 2 to execute value crossover operation with parent1. The detail of the topology crossover is explained in Sect. 4.3.

Step 6: Value crossover—The next step is value crossover. Two parents are selected by n-tournament selection. Two function nodes will be selected from the parents respectively and go through value crossover with probability $P_{valuecross}$. The resulting child node of the value crossover will be taken as the corresponding function node of Parent 1, replacing Parent 1's previous function node. This crossover operator is introduced specifically in Sect. 4.4.

Step 7: Mutation—The last step is mutation. One of two mutation operations (delete or add) is randomly selected with equal probability to being executed on the *i-th* individual. The mutation operators are introduced in Sect. 3.1.

4.2 Population initialization

Population initialization defines how the individuals in the initial population are generated and how to ensure that their structure is valid. Each tree individual is generated by the *grow* method [49], which means that the distances between each leaf nodes and the root node are not the same.

The three types of structure check mentioned in Sect. 3.2 are applied to prevent invalid circuits.

4.3 Circuit topology evolution

Based on the proposed circuit representation, circuit topology evolution is implemented by executing the operations of topology crossover and mutation. Figure 8 shows an example of topology crossover operation based on the proposed circuit representation. The crossover operation is executed among non-leaf nodes. As shown in Fig. 8, two parents are selected among T individuals by n-tournament strategy. A node of the first parent and a node of the second parent are randomly selected, e.g., terminal node 2 and node C_1 in Fig. 8 and node C_1 of Parent 2 is also randomly selected. Then the sub-tree rooted at the selected node from the second

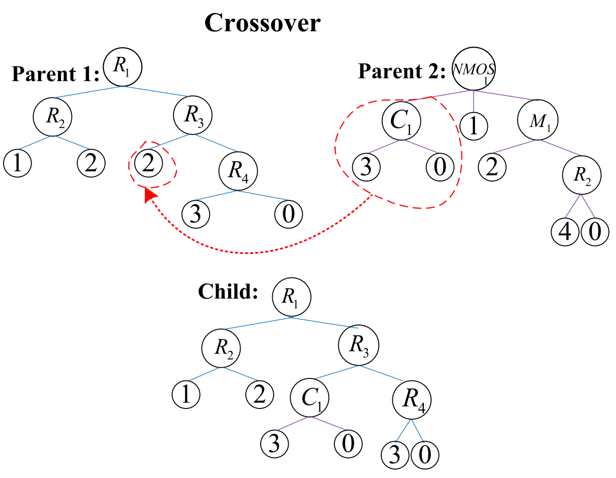


Fig. 8 An example of topology crossover operation

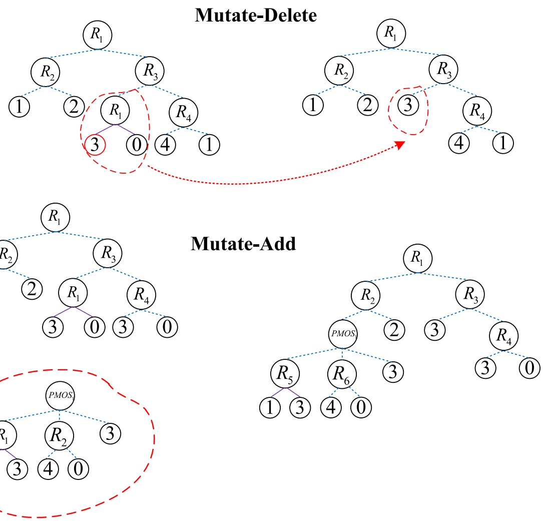


Fig. 9 An example of mutation operation

parent is used to replace the sub-tree rooted at the selected node from the first parent to generate a child, which will be added into the population.

Figure 9 shows an example of mutation operation that explains how the mutation operation is executed based on proposed circuit representation. Two types of mutation operations are applied in this work, which are *delete* and *add*, respectively. As for the *delete* operator, a subtree is randomly selected to be deleted and the left-most terminal node of this sub-tree is used to replace it. As for the *add* operator, a terminal node from the current tree is selected uniformly at random, and then a randomly generated sub-tree with a feasible depth is created to replace the selected terminal node. “Feasible depth” here means that its depth must be between the minimum and maximum depth parameter of the algorithm. This is implemented by the three types of structure checks introduced in Sect. 3.2. The upper part of Fig. 9 shows an example of *delete* operation, in which the function node R_1 is randomly selected to be deleted and its left terminal node *3* will substitute the node R_1 itself. The bottom part of Fig. 9 shows an example of *add* operation, in which a random terminal node *1* is substituted by a newly generated sub-tree.

4.4 Device value optimization

Besides the circuit topology, the device values of the circuits also need to be determined automatically for the circuit design. In this work, the device value is represented by the value trees (proposed in Sect. 3.1), which are embedded in the corresponding function nodes. The device value is evolved by the operation of value crossover alongside the evolution of the circuit topology as outlined in the flowchart from Fig. 7. The value crossover operation can only be executed to two function nodes of the same type of device. Here, the “same type" refers to the same device type of function nodes. For example, the value of a resistor could only go through value crossover with the value of another node that also represents a resistor. Similarly, a node representing a capacitor cannot go through value crossover with another node representing a resistor.

Figure 10 shows the detail that how the value crossover works. R_1 is a randomly selected function node in the *i-th* individual. R_2 is a randomly selected function node of Parent 1 (which was selected by tournament selection) among the function nodes of the same type as R_1. If no function node of the same type exists, then another function node could be selected until there is the one with the same type. Assume that both nodes correspond to a resistor. Their corresponding value trees go through crossover as follows. A sub-tree is randomly selected for each of the two value trees. The sub-tree from the value tree of R_2 then replaces the sub-tree from the value

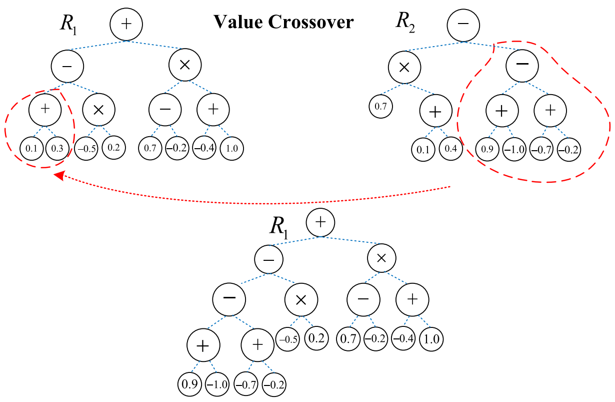


Fig. 10 An example of value crossover for a resistor

Table 2 GP-based circuit evolutionary algorithm parameters

Parameters	Value
fitness	Depends on cases
P	100
Representation	Proposed method
M	800
P_{cross}	0.8
$P_{valuecross}$	0.2
P_{mutate}	0.2
Parent selection	Tournament(T=20)

tree of R_1. In this example, after the value crossover is applied, the value tree of R_1 changes from $169.8K\Omega$ to $70.8K\Omega$ according to Eq. (1).

As mentioned above, evolutionary design of analog circuit contains both of circuit topology evolution and device value optimization, which allows for the diversity of evolved circuits and better evolution results. The specific genetic operators (crossover and mutation) are proposed for the circuit topology evolution and device value optimization. The sub-circuits of a whole circuit can be represented by the branches of a tree. Therefore, the crossover operation executed on the tree stands for the exchanging between sub-circuits. This is an advantage over graph-based representations, for which suitable crossover operators are difficult to design. Mutation operations include delete and add, which can increase and reduce the depth of a tree, allowing for increasing the diversity of circuit topology.

5 Experimental studies

We implemented our GP in Python. We also use Python to generate netlists. The implementations will be made available as open source in GitHub.[1] The performance of the evolved circuits is evaluated in simulation using NGSPICE [50], which is based on Spice3 [51].

Three benchmark circuits are chosen to evaluate the proposed approach, namely voltage reference circuit, temperature sensor circuit, and Gaussian function generator, respectively. These benchmark circuits are widely applied to evaluate the automated circuit design methods [13, 14, 19]. In this section, we show how our proposed method can be applied to evolve these three benchmark circuits and compare the results with existing approaches. The circuit evaluation of all the compared work are carried on the NGSPICE. In order to make a fair comparison, all the fitness values are computed by the same fitness function proposed in [19], and will be explained in Sect. 5.1. Moreover, the related parameters are listed in Table 2.

[1] https://github.com/embeddedsky/EvoCkt.git.

5.1 Experimental setup

5.1.1 Experimental setup for voltage reference circuit

The first experiment is to evolve a voltage reference circuit, which is to produce a fixed output voltage $V_{out} = 2V$ on the load resistor when the input voltage varies within the interval $4V \leq V_i \leq 6V$ and the circuit temperature varies within the interval 0 °C $\leq T \leq$ 100 °C $\leq$ 100 °C. All the candidate circuits are simulated with DC sweep, where the intervals of the input voltage DC sweep are $0.1V$ and the intervals of the temperature sweep are 25 °C. Therefore, there will be 21 discrete values of input voltage and 5 discrete values of temperature, giving a total of 105 measured points for the DC sweep simulation.

- Embryo circuit: The embryo circuit of voltage reference circuit is shown in Fig. 11. There are three accessible nodes, which is one power supply (with a $1K\Omega$ input resistor), one output load resistor ($10K\Omega$), and ground. The part marked by the dashed line will be further evolved.
- Fitness function: Under the different circuit temperatures T_i, each measured output voltage point $V_{outi,j}$ has its corresponding target value $V^*_{outi,j}$, where i denotes the i-th circuit temperature, and j denotes the sampled points of output voltage. The fitness function is defined as the following equation [19]:

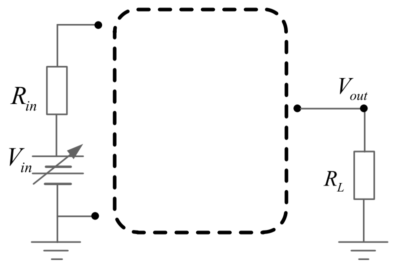
$$fitness = -\sum_{i,j} \varepsilon_{ij}, \tag{3}$$

where ε_{ij} is :

$$\varepsilon_{ij} = \begin{cases} \left(V_{outi,j} - V^*_{outi,j}\right)^2, \text{if } |V_{outi,j} - V^*_{outi,j}| \geq 0.01V \\ 0, \qquad\qquad\qquad \text{if } |V_{outi,j} - V^*_{outi,j}| < 0.01V. \end{cases} \tag{4}$$

- Device set: The devices used for evolving the voltage reference circuit are the same as those used by the baseline approaches, namely NPN (BC846B), PNP (BC856B) bipolar junction transistor, and resistors.
- Circuit Quality Measurement: "Hit" is applied in this work to measure the evolved circuit quality, which has been widely used in previous work [13, 14,

Fig. 11 The diagram of embryo circuit for voltage reference circuit. Where $R_{in} = 1K\Omega$, load resistor $R_L = 10K\Omega$, and the output voltage V_{out} will be measured to be evaluated



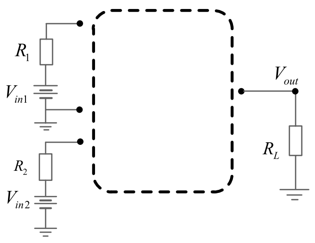


Fig. 12 The diagram of embryo circuit for temperature sensor circuit. Where $V_{in1} = 15V$, $V_{in2} = 5V$, $R_1 = 1K\Omega$, $R_2 = 1K\Omega$, load resistor $R_L = 10K\Omega$, and the output voltage V_{out} will be measured to be evaluated

19]. "Hit" refers to the situation where the absolute difference between the measured point of output voltage and its target value (*error*) is less than or equal to $0.02V$. Therefore, the proportion of the number of "Hit" to the total number of measured points indicates the quality of the evolved circuit.

5.1.2 Experimental setup for temperature sensor circuit

The second evaluation task is to evolve a temperature sensor circuit, of which output voltage is varying with the different circuit temperatures. The variation range of temperature is 0 °C $\leq T \leq$ 100 °C, and the variation range of output voltage is $0V \leq V_i \leq 6V$. All of the candidate circuits will be simulated with temperature sweep. The sample step of temperature is 5 °C , giving 21 measured points for the sweep simulation.

- Embryo circuit: The embryo circuit for evolving temperature sensor circuit is shown in Fig. 12. There are four accessible nodes, where one is the positive voltage supply (V_{in1} with a series resistor R_1), one is the negative voltage supply (V_{in2} with a series resistor R_2), one is the output terminal (load resistor R_L), and final one is ground. The part marked by the dashed line is to-be evolved circuit.
- Fitness function: At the different circuit temperatures T_i, the output voltage V_{out} is different, which will be measured to be evaluated. The *i-th* target value of output voltage is linear-related to the circuit temperature T_i, which is defined as $V^*_{outi} = \eta T_i$. η is a constant representing the linear relation between circuit temperature and output voltage. The fitness function is defined as following [19]:

$$fitness = -\sum_i \left(V_{outi} - V^*_{outi}\right)^2. \tag{5}$$

- Device Set: The devices applied to evolving temperature sensor circuit are the same as those used for evolving voltage reference circuit mentioned in Sect. 5.1.1.
- Circuit quality measurement: Different from voltage reference circuit, the standard of "Hit" for temperature sensor circuit is that the absolute difference between the measured output voltage and target (*error*) is less than or equal to $0.1V$, which is the same as the one has been set in [13]. The rate of "Hit" will indicate the evolved circuit quality.

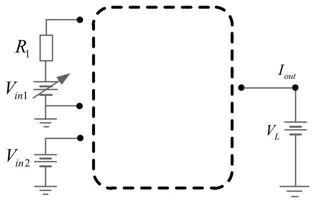


Fig. 13 The diagram of embryo circuit for Gaussian function generator. Where $2V \leq V_{in1} \leq 3V$, $V_{in2} = 5V$, $R_1 = 1\Omega$, and $V_L = 2.5V$. I_{out} will be measured to be evaluated

5.1.3 Experimental setup for Gaussian function generator

The final task is to evolve a Gaussian function generator, of which output current is a Gaussian function of input voltage. All of the candidate circuits will be simulated with DC sweep, where the variation range of input voltage is $2V \leq V_{in1} \leq 3V$. The target output current is the Gaussian function with a peak value $I_{out}^{max} = 80nA$ in correspondence of $V_{in1} = 2.5V$ and sweep step is $10mV$, which provides a total of 101 measured points for DC sweep simulation.

- Embryo circuit: The embryo circuit for evolving Gaussian function generator is shown in Fig. 13 There are four accessible nodes, where one is the variable voltage supply (V_{in1} with series resistor R_1), one is the fixed voltage supply (V_{in2}), one is the output terminal (with another voltage supply V_L), and the final one is ground. The part marked by the dashed line will be further evolved.
- Fitness function: During the circuit evolution, the output current will be measured to be evaluated. There will be the different target values of output current corresponding to the different input voltage V_{in1}. Therefore, the fitness function is defined as follows [19]:

$$fitness = -10^{14} * \sum_i \left(I_{outi} - I_{outi}^*\right)^2. \tag{6}$$

  where I_{outi} is the measured current and I_{outi}^* is the corresponding target value. The value -10^{14} is a factor to normalize the square of the error, since the unit of swept current is nano-level, and the square of the differences between target current and measured current will be much smaller.
- Device set: MOSFETs are applied to evolving Gaussian function generator and resistors are also necessary for Gaussian function generator. The type and model of the devices used in this work are the same as the baseline approaches [14] and [13].
- Circuit quality measurement: As for the Gaussian function generator, there is also minor change for the standard of “Hit". The absolute difference between measured current I_{outi} and target value I_{outi}^* ($error$) is less than or equal to $5nA$, a “Hit" will be scored. The circuit quality measurement applied in this work is the same as the one used in [13].

Table 3 Results for the experiments with different the number of evaluations

Tasks	Evaluations	BF*	MBF±std
Voltage reference	2×10^4	−0.0228	−0.8141±1.0961
	5×10^4	−0.0081	−0.2594±0.7815
	8×10^4	−0.0070	−0.0319±0.0231
Temperature sensor	2×10^4	−0.0093	−0.1941±0.2058
Temperature sensor	5×10^4	−0.0089	−0.0502±0.0480
Temperature sensor	8×10^4	−0.0084	−0.0233±0.0232
Gaussian function	2×10^4	−0.0158	−0.2865±0.2472
	5×10^4	−0.0074	−0.1104±0.0944
	8×10^4	−0.0063	−0.0883±0.0862

*Fitness is with negative sign, so the best fitness (BF) is to maximize the negative fitness value

5.2 Experiment results

Based on the experimental setup mentioned above, this section gives the validation results of the proposed method. The fitness evaluation results for three benchmark circuits, the benefits of the proposed crossover operators and structure check, evolved circuits and result comparisons with existing literature will be introduced in following Sects. 5.2.1, 5.2.2, 5.2.3 and 5.2.4, respectively.

5.2.1 Fitness evaluation results

This section gives the fitness evaluation results for each benchmark circuit. Several metrics are used for a statistical evaluation of the proposed method, such as BF and MBF, which have been widely used to measure the performance of evolutionary algorithms [13, 52]. BF defines the best fitness obtained in 20 runs and MBF is the acronym of Mean Best Fitness, which is the average value of the best fitness obtained in each run. Table 3 shows the results for the experiments with different number of evaluations. As Table 3 shows, with the increase of the number of fitness evaluations from 2×10^4 to 8×10^4, BF and MBF are more closing to 0, as expected. The average fitness and error curves for 20 runs of the experiments of each benchmark circuit will be given in the appendix.

5.2.2 Validating the benefits of our proposed tree-Based crossover operators and feasibility checks

As explained in Sect. 3.3, our tree-based representation lends itself to a suitable crossover operator. In particular, our topology crossover stands for exchanging sub-circuits between individuals. This may lead to better fitness than algorithmic designs that do not make use of crossover, being an advantage over algorithm designs such as graph-based designs. Therefore, we evaluate whether the

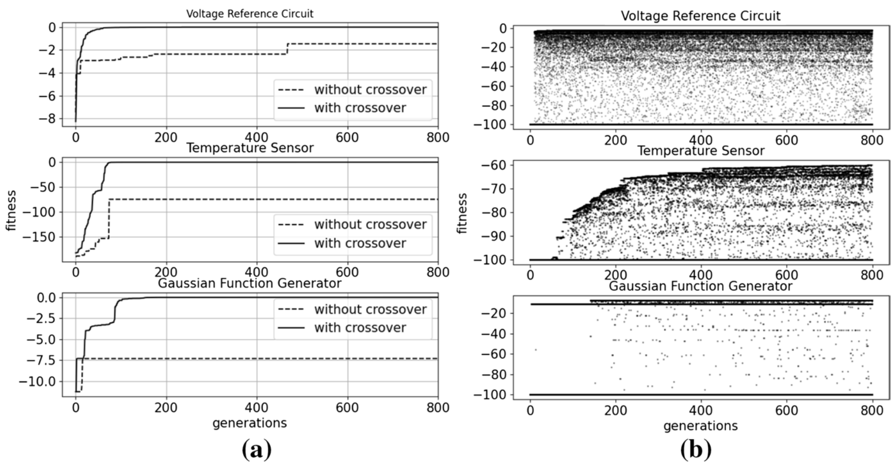


Fig. 14 **a** The average fitness comparisons with crossover and without crossover operation for different benchmark circuits. **b** Fitness distribution without structure check for different benchmark circuit

introduction of our crossover operators is helpful to improve fitness. Figure 14a shows the fitness across generations when applying and when not applying the crossover operators. We can see that the evolution with crossover operators leads to better fitness results than the one without crossover operators, confirming the benefits of our tree-based representation, which lends itself to the adoption of crossover operators.

In addition, the effectiveness of the structure check proposed in Sect. 3.2 is verified by executing the experiments without the structure check, where the infeasible individuals are given a penalty instead of applying the proposed structure check to revise them. Several approaches in the literature are based on penalties, such as Gan's work [17]. As Fig. 14b shows, there are a lot of infeasible individuals ($fitness = -100$) in the population, which may limit the diversity of the population incurring worse fitness.

5.2.3 Evolved circuits

Besides the analysis of the evaluation fitness, the evolved results also play an important part for verifying the proposed algorithm. Figure 15 shows the diagram of measured output of the best evolved voltage reference circuit (red lines) and its ideal output (green line), and Fig. 16 gives its circuit scheme. Correspondingly, Figs. 17, 18, 19 and 20 show the output voltage of temperature sensor and Gaussian generator circuits, as well their circuit schemes, respectively. According to the Figs. 15, 16, 17, 18, 19 and 20, we can see that the evolved circuits do not have infeasible structures and can be simulated successfully. Moreover, the output signals of evolved circuits follows the target outputs well.

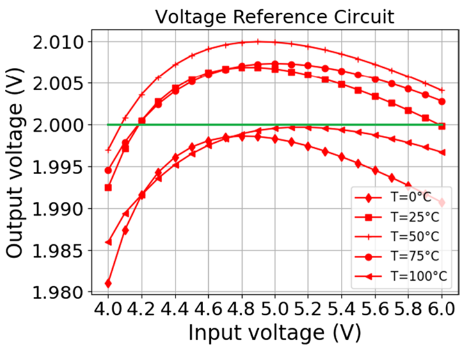
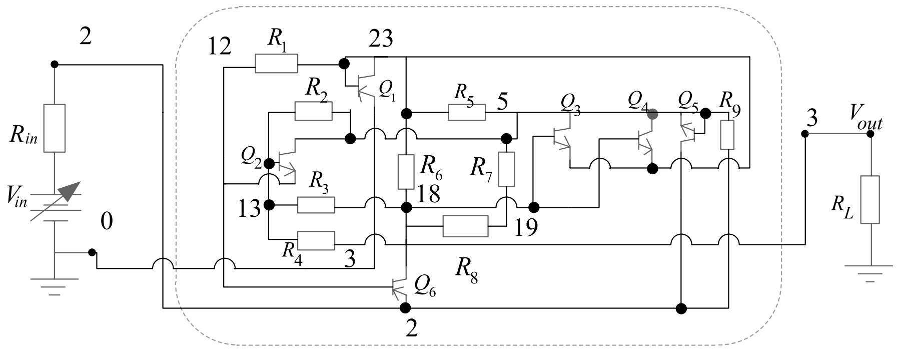


Fig. 15 The diagram of measured output of the best evolved voltage reference circuit (red lines) and its ideal output (green line)



$$R_1 = 16954.5\Omega \quad R_4 = 7.2\Omega \quad R_7 = 6057.7\Omega$$
$$R_2 = 461.2\Omega \quad R_5 = 13757.5\Omega \quad R_8 = 2357.2\Omega$$
$$R_3 = 200000\Omega \quad R_6 = 382.6\Omega \quad R_9 = 2.7\Omega$$

Fig. 16 The best evolved results of voltage reference circuit

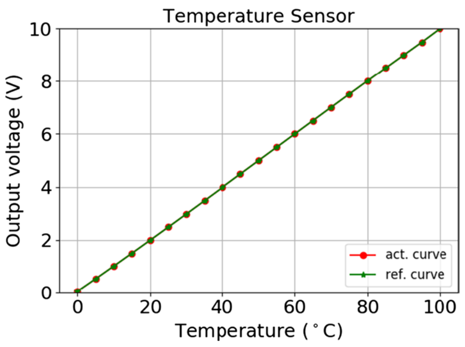
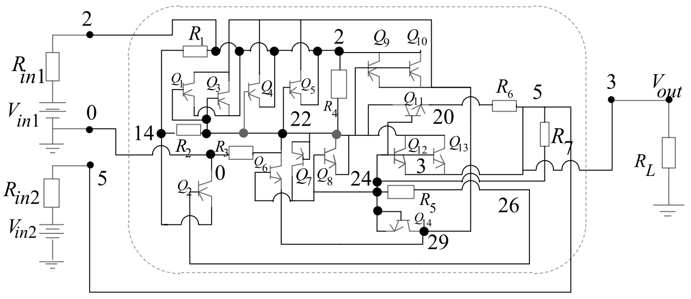


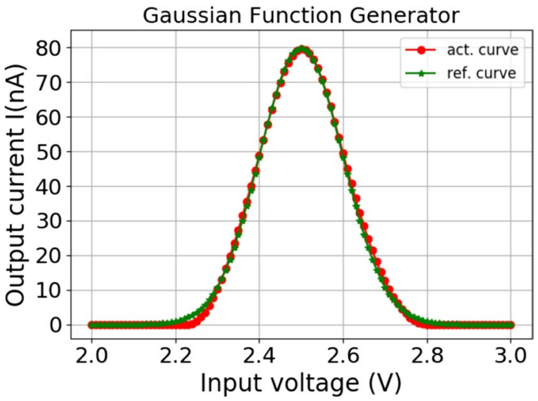
Fig. 17 The diagram of measured output (red line with circle) of the best evolved temperature sensor circuit and its reference target (green line with triangle)

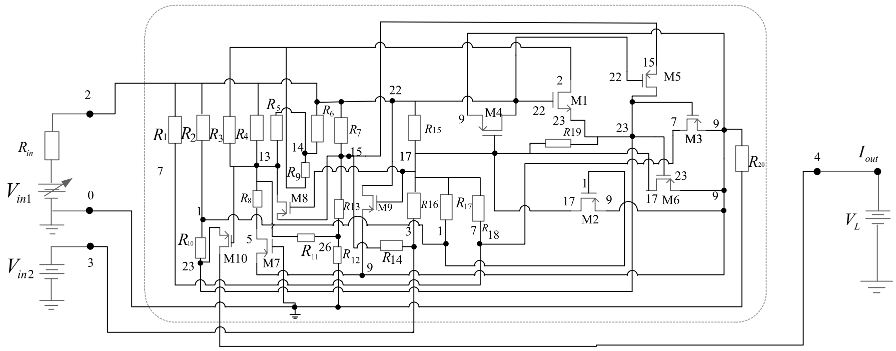


$R_1 = 1565.3\Omega$ $R_4 = 50.5\Omega$ $R_7 = 127614.9\Omega$
$R_2 = 6.9\Omega$ $R_5 = 200000\Omega$
$R_3 = 18985.9\Omega$ $R_6 = 6617.2\Omega$

Fig. 18 The best evolved results of temperature sensor circuit

Fig. 19 The diagram of measured output (red line with circle) of the best evolved Gaussian function generator and its reference target (green line with triangle)





$R_1 = 3624.1\Omega$ $R_4 = 14412.9\Omega$ $R_7 = 109958.7\Omega$ $R_{10} = 6389.2\Omega$ $R_{13} = 6868.7\Omega$ $R_{16} = 1000\Omega$ $R_{19} = 11664.2\Omega$
$R_2 = 922.5\Omega$ $R_5 = 10069.9\Omega$ $R_8 = 2357.2\Omega$ $R_{11} = 500\Omega$ $R_{14} = 11735.7\Omega$ $R_{17} = 1979.1\Omega$ $R_{20} = 1258.2\Omega$
$R_3 = 7175.7\Omega$ $R_6 = 38163.2\Omega$ $R_9 = 5761.1\Omega$ $R_{12} = 470185.6\Omega$ $R_{15} = 500\Omega$ $R_{18} = 18332.6\Omega$

Fig. 20 The best evolved results of Gaussian function generator

Table 4 Comparisons with previous work for the three benchmark circuits

Parameters	Koza's [19]	Matthiussi's [14]	Federico's [13]	Ours
Vol. reference				
Absolute value of fitness	6.6	2.64	0.112	**0.0233**
Evaluations	5.12×10^7	5.6×10^6	1.86×10^6	$\mathbf{8 \times 10^4}$
Hits/max	89.9/105	98.1/105	70.7/105	94/105
Components	67	70.2	32	**15**
Tem. sensor				
Absolute value of fitness	26.4	1.13	0.065	0.0883
Evaluations	1.6×10^7	6.5×10^6	6.14×10^6	$\mathbf{8 \times 10^4}$
Hits/max	16/21	20.3/21	19.9/21	19.1/21
Components	54	27.8	33	**21**
Gau. function				
Absolute value of fitness	0.094	0.3	0.036	**0.0319**
Evaluations	2.3×10^7	4.3×10^6	6.23×10^6	$\mathbf{8 \times 10^4}$
Hits/max	101/101	98.3/101	85.0/101	**99.6/101**
Components	14	36	28	30

5.2.4 Comparison with existing approaches

To further validate the proposed method, this section compares its results with those of the previous approaches, which are shown in Table 4. The data shown in the last column of the Table 4 are the average results for 20 runs of the proposed method. As mentioned in [14], despite the differences in the number of runs in other's work, it is useful to consider side by side the results obtained with those methods.

As for the results of evolving voltage reference circuit, the proposed method produces better fitness (0.0233) with less number of evaluations (8×10^4), where the fitness of other work are 6.6 (Koza's [19]), 2.64 (Mattiusssi's [14]) and 0.112 (Federico's [13]), respectively. It is worth to note that the fitness of proposed work is much better than that of Koza's [19] although the two methods are all based on tree structure. This highlights the advantage of our proposed tree representation. Hit rate of the proposed method is greater than the work presented by Koza [19] and Federico [13]. Moreover, our evolved voltage reference circuit has fewer components than all others.

In the case of temperature sensor circuit, the fitness produced by Federico's method is slightly better than the one produced by our method, but using about two orders of magnitude more circuit evaluations. Moreover, the result temperature sensor circuit evolved by our method contains the least number of components compared with all other methods.

In the case of Gaussian function generator, the proposed method also can produce better fitness (0.0319) with less number of evaluations (8×10^4). In addition, the hit rate of proposed method is higher than other methods. And also the evolved results of temperature sensor circuit have fewer components, which is more compact than

Table 5 Comparisons with manual-designed circuits

	Work	Implementation	#Components
Vol. reference	Ours	BJT+resistor	**15**
	Manual design [53]	BJT + resistor + capacitor	29
Tem. sensor	Ours	BJT + resistor	**21**
	Manual design [54]	BJT + resistor + diode	23
Gau. function	Ours	MOSFET + resistor	**30**
	Manual design [55]	MOSFET	30

the left three. And the hit rate of proposed method is also competitive with other works. The circuit evolved by our proposed method is more compact than the one proposed by Matthiussi [14].

As it can be seen in the above-mentioned experiment results, the feasibility of our proposed approach has been verified considering the evolved circuit performance and the fitness comparisons with previous work. In addition, our method usually improves the fitness results while using less number of evaluations, achieves a competitive hit rate, and usually uses fewer components to construct the circuits.

As for the manual design of the benchmark circuits, engineers have proposed different ways to construct these circuits, while the strong experience and circuit knowledge are highly required. Some classic manual-designed circuits for benchmark circuits are applied to compare with those of our approach evolved. Paul et al. [53] proposed a voltage reference circuit, which is composed of 15 BJTs, 13 resistors and 1 capacitor. Meijer et al. [54] proposed a temperature sensor, which is composed of 8 BJTs, 7 diodes, 2 capacitors and 6 resistors. Popa [55] proposed a Gaussian function generator that consists of 30 Mosfets. The comparison with manual-designed circuits is listed in Table 5. Compared with these manual designs, our proposed method provides an automated design tool for designing the analog circuits without the high requirement of circuit experience and knowledge, and the evolved circuits are human-competitive, which not only can realize their corresponding functions but also have compact size.

We have also inferred that why our proposed method outperformed other's work is fused by two parts. Firstly, sub-circuits of a whole circuit could be represented by the branches of a tree directly, therefore, the crossover operation executed on the tree stands for the exchanging between sub-circuits, which is beneficial for the evolution process as shown in Sect. 5.2.2. In Koza's tree [11], the sub-circuits are determined by not only the corresponding branches but also the other function nodes located in its parent's node level. As a result, their crossover operations between the branches may miss the corresponding function nodes in its parent's node level, which will be disruptive to evolution process. Secondly, because of three types of structure check for circuit, each circuit individual decoded by our tree-based representation could be simulated and evaluated, preventing the problem of infeasible chromosomes in Federico's approach [13]. In Federico's work [13], the circuit individual that cannot be simulated could be still generated, though they will are strongly penalized.

As shown in Sect. 5.2.2, the structural checks adopted by our approach are more efficient than the use of a penalty. In addition, according to circuit simulation results, our evolved circuits can ensure the circuit feasibility successfully.

6 Evolving a memristor-based pulse generation circuit

With the rapid development of microelectronics and semiconductor technology, Moore's Law is breaking down. Device size and power consumption have become critical for developing electronic technologies. Therefore, memristor research has been widely spread in the field of electronic technology for its nano-size and energy efficiency [56]. In addition, memristors also have nonvolatility and resistance variability [46], which can potentially be useful for many applications such as neuromorphic computing, storage and logic computation.

Most neuromorphic circuits are based on spike or pulse, having similarities with the neural system [23]. Therefore, it is necessary for researchers to study the circuit counterpart of pulse generators in order to adopt neuromorphic computing. For realizing pulse generation circuits in neuromorphic computing, some researchers relied on external chips to generate pulse [22, 24]. However, this leads to bulk control or auxiliary circuits. Moreover, applying external chips as pulse generation circuits also has the problem of inefficient resource utilization of chips. Therefore, some researchers have made attempts to design the in-chip pulse generation circuits with traditional analog devices to prevent using external chips [57, 58]. However, based on the experience of circuit designers, some of these circuits contain capacitors, which will occupy most of area of the pulse generation circuits [59]. Besides, the circuits contain large number of traditional devices such as MOSFETs, which may also lead to large size and high power consumption.

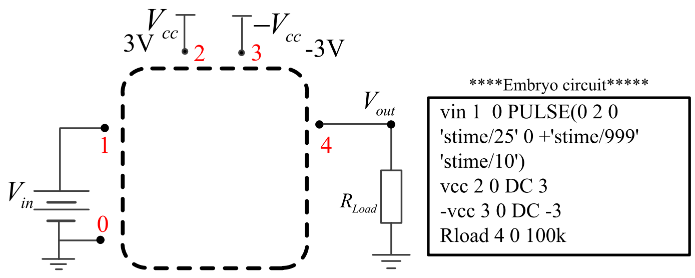
Therefore, it is desirable to use memristors themselves to construct pulse generation for neuromorphic computing, due to their nano-size, energy efficiency and dynamic behavior. However, due to the complex dynamic behaviors of memristors and comprehensive circuit requirements for neuromorphic computing, it is time-consuming for designers to design a memristor-based pulse generation circuit with high quality. Therefore, it is valuable to apply our proposed method to evolve a memristor-based pulse generation circuit.

In this section, the proposed method will be applied to evolving a memristor-based pulse generation circuit and compared against similar purpose circuits designed manually [22, 24, 57, 58, 60].

6.1 Experimental setup

6.1.1 Memristor model

In order to evolve a memristor-based pulse generation circuit, the selection of memristor models is the first consideration. Since the invention of HP TiO_2 memristor [56], some research groups pay their attention to modelling memristor



```
vin 1  0 PULSE(0 2 0
'stime/25' 0 +'stime/999'
'stime/10')
vcc 2 0 DC 3
-vcc 3 0 DC -3
Rload 4 0 100k
```

Fig. 21 Schematic diagram and netlist of embryo circuit for evolving memristor-based pulse generation circuit

dynamic behaviors into formulas [61, 62]. HP model is a classic memristor device model introduced by Strukov et al. [56], where a time-domain differential equation was proposed to describe the physical behavior of linear ion drift in a memristor [63]. Therefore, in this work, Hewlett–Packard (HP) TiO_2 memristor model is adopted for the concept verification in our design and simulation.

Since the invention of HP TiO_2 memristor [56], some research groups pay their attention to modelling memristor dynamic behaviors into formulas. Volt ampere characteristics of memristor can be described by the following algebraic equation [56]:

$$v(t) = \left[R_{on}\frac{w(t)}{D} + R_{off}\left(1 - \frac{w(t)}{D}\right)\right]i(t), \tag{7}$$

where D denotes the total length of memristor; R_{on} and R_{off} represent the resistances of oxygen vacancy and oxygen-deficient vacancy parts of memristor, respectively; $w(t)$ is the oxygen vacancy part; and d$v(t)$ and $i(t)$ are the voltage applied on memristor and the current flowing across the memristor, respectively.

The state variable of memristor changes with the applied external signal, which can be described by the following differential equation:

$$\frac{dx(t)}{dt} = \frac{\mu_v}{D^2}i(t)f(x(t)), \tag{8}$$

where μ_v is the dopant mobility of the material and $x(t)$, which is regarded as the state variable of memristor, is the proportion of the length of the oxygen vacancy part $w(t)$ and total length of memristor D. $f(x(t))$ is a window function to get over the boundary effect of memristor [64], which can be described as follows:

$$f(x(t)) = 1 - (2x(t) - 1)^{2p}, \tag{9}$$

where $p \in Z_+$ is a parameter of controlling the non-linear degree of the window function.

Table 6 Circuit evolutionary algorithm parameter setting

Parameters	Value
Number of generations	1000
Population size	100
Crossover rate	0.6
Mutation rate	0.2
Value crossover rate	0.6
Tree depth	4–7
Tournament size	20

6.1.2 Circuit configuration and evolutionary algorithm

In order to evolve the pulse generation circuit, a proper embryo circuit should be predefined. As shown in Fig. 21, five circuit components are predefined in the netlist, which are input source V_{in}, pull-up voltage V_{cc}, pull-off voltage $-V_{cc}$, ground port and the load R_{load}, respectively. And their position information (0, 1, 2, 3 ,4) will be added into the terminal node set in advance.

Besides setting up the embryo circuit, the following items also need to be set up:

- Input and target output: Similar with the neuronal dynamic behaviors, the output pulse of pulse generator can be triggered by accumulated potential. Therefore, the accumulated potential will be set as the input of the evolved circuit. For simplification, the saw-tooth wave with 2 V amplitude and $1ns$ period will be used as input in this work. The square pulse is the most commonly generated by pulse generation circuits, and will thus be set as the target output, of which the amplitude is $2V$ and period is 1 ns.
- Set of devices: To our best knowledge, there is no pulse generation circuit that is only composed of memristors, even in the logical circuit content [65]. Memristors and MOSFET are commonly used for constructing pulse generation circuits [60, 66]. Therefore, in our design, in order to manipulate the dynamic switching behaviors of memristor sufficiently, MOSFET, resistors, and memristors are added into the devices set.
- Simulation options: Combined with the rate of state switching of memristor, the simulation time is set as $10ns$. To trade off the execution time and sample accuracy, the number of sample points of the target output of the circuit is set as 1000. We need the variation of output with time; therefore, the simulation type is the transient analysis.

Besides the circuit configuration, parameters of the algorithms were chosen based on preliminary experiments, and are listed in Table 6. Besides, the fitness function of evolving the memristor-based pulse generation circuit is defined as follows:

Table 7 Results for evolving memristor-based pulse generation circuits with different the number of evaluations

#Evaluations	BF*	MBF ± std
2×10^4	– 16.84	– 37.32 ± 16.54
4×10^4	– 12.45	– 31.28 ± 19.02
6×10^4	– 11.46	– 22.01 ± 12.23
8×10^4	– 10.69	– 15.02 ± 4.54
10×10^5	– 10.47	– 12.91 ± 2.10

* Fitness is with negative sign, therefore, the best fitness (BF) is to maximize the negative fitness value

$$fitness = -100 * \frac{\sum_i \left| V_{outi} - V^*_{outi} \right|}{N}, \tag{10}$$

where N is the total number of sampled points, which is set as 1000; V_{outi} is the *i-th* sampled point of output voltage and V^*_{outi} is its corresponding target value.

6.2 Experiment results

The results for the experiments with different number of evaluations are shown in Table 7. As it shown in Table 7, the BF (Best fitness for one run) and MBF (Mean Best Fitness) across the 20 runs improve with the increase of the evaluations. Figure 24 in the appendix shows the average fitness and error curves for 20 runs of the experiments.

Figure 22 shows the input, output and target voltage of the evolved memristor-based pulse generation circuit. As we can see, under the triangle input voltage, the evolved circuit can generate a regular pulse, which is highly matched to the target curve. In order to qualify the output performance of the evolved pulse generation circuit, mean-square error (MSE), which has been widely applied for evaluating the differences between the actual and target output of circuit [67], is applied for measuring the quality of the circuit output in this work.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(V^*_{outi} - V_{outi}\right)^2, \tag{11}$$

where V^*_{outi}, V_{outi} are target voltage value, actual output voltage value for a given data point i, respectively. N is the number of sampled points of the output voltage. MSE of the result shown in Fig. 22 is 0.15, which meets the requirement of the circuit with similar purpose proposed in [22]. Figure 23 shows the evolved memristor-based pulse generation circuit, including the evolved resistance of the resistors contained in the circuit. The circuit contains 8 memristors, 6 resistors and 10 transistors. The initial state of all the memristors in the evolved circuit is OFF.

Table 9 gives the comparisons of the evolved circuit and other manual design circuits with similar purpose. To validate the proposed approach, we compare the

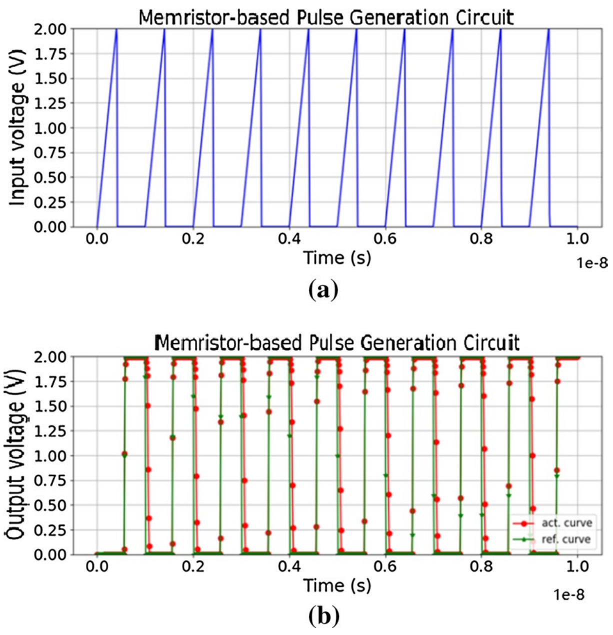


Fig. 22 Saw-tooth wave input and the circuit simulation results of the best evolved circuit. **a** Saw-tooth wave input. **b** Target voltage

works from different perspectives, which are the usage of external chip, the ways of realizing, the number of transistors and capacitors, area and energy. According to Wu et al. [68], the energy dissipation per pulse can be defined as the differences between the input power consumption P_{IN} and output power consumption P_{OUT} during the duration of one pulse, where P_{IN} and P_{OUT} can be calculated by the following equation:

$$P = \int_0^T |V(t) \cdot I(t)| dt. \tag{12}$$

where $V(t)$ and $I(t)$ are the voltage and current at the sampled points.

In addition, the area of the single MOSFET can be calculated by the following equations:

$$AREA_{MOS} = W \times L \times k. \tag{13}$$

where W and L refer to the width and length of the channel, and k is a factor showing how much bigger a transistor is than its channel area. As for the area of one HP memristor, it is set as 9 nm^2, as proposed in previous work [69, 70]. Considering the compatibility of memristor and MOSFET, 45nm CMOS technology is applied in this work for MOSFET and resistor. The area of one resistor is set as $2025nm^2$ ($45nm \times 45nm$). The area of one MOSFET is set as $0.6075\mu m^2$($450nm \times 450nm \times 3$). Our current estimation of power and area of circuits may

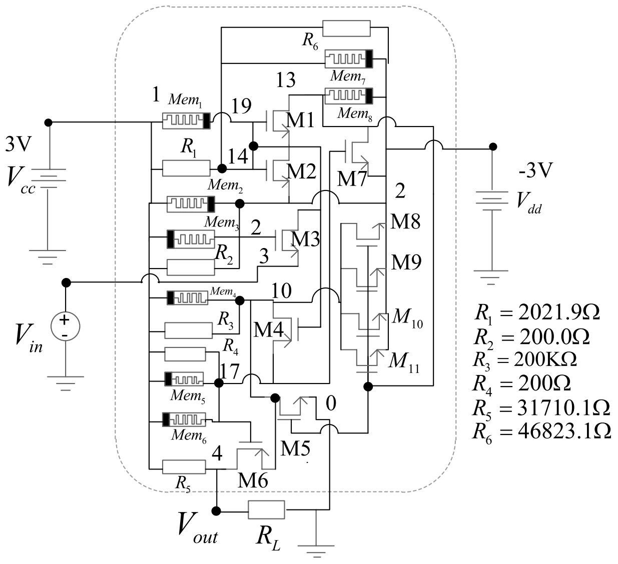


Fig. 23 The best evolved results of voltage reference circuit

Table 8 Area and value range of the devices

Devices	Memristor	MOSFET*	Resistor
Area	$9nm^2$	$0.6075\mu m^2$	2025 nm^2
Value range	OFF or ON	–	200–200K

* The area of one MOSFET is calculated by Eq. (13)

Table 9 Comparisons of the evolved memristive pulse generation circuit and other works

Works	External chip?	Realizing method	#Transistors	#Capacitors	Area	Energy/pulse (nJ)
[22]	YES	555 timer	25	2	9 mm^2	240
[24]	YES	555 timer	25	2	9 mm^2	240
[57]	NO	Transistor+Capacitor	19	3	18,340 μm^2	4500
[58]	NO	Transistor+Capacitor	20	1	1705 μm^2	2.85
[60]	NO	Transistor+Memristor	16	1	23.24 μm^2	125
[66]	NO	Transistor+Memristor	25	0	46.37 μm^2	–
Ours	NO	Transistor+Memristor	**10**	**0**	**6.08**μm^2	**1.53**

not be accurate and could be improved in future work. Table 8 shows the area and device value range of the devices to be used in evolving circuits. The initial states (ON or OFF) of different memristors and the specific value of different resistors will be determined by the proposed algorithm.

As shown in Table 9, some researchers applied an external chip to generate pulse, 555 timer, which contains 25 transistors and 2 capacitors, leading to large circuit area and power consumption [22, 24]. Petit et al. used 19 transistors and 3 capacitors to implement the pulse generation, which occupies $18340\mu m^2$ circuit area and generates 4500nJ energy for one pulse [57]. In the work of Wijekoon et al. [71], 20 transistors and 1 capacitor were applied to design the circuit for generating pulse, of which area is $1705\mu m^2$ and energy dissipated per pulse is $2.85nJ$.

Compared with the work of [22, 24], our evolutionary approach prevents applying the external chips as pulse generator, which solves the problems of bulk auxiliary circuits and inefficient resource utilization of chips. Compared with the manual design circuits proposed by [57, 71], the evolved memristor-based pulse generation circuit is equipped with fewer transistors (0 capacitor), lower energy consumption, and more compact design, which is better for neuromorphic computing. The work proposed in [60, 66] also applied memristors to implement pulse generation. As for work [60], additional 16 transistors and 1 capacitor were used. As for work [66], 25 MOSFETS are applied extra besides the memristors, which were more than the ones evolved by our proposed method (10 transistors and 0 capacitor).

7 Conclusion

In this paper, a novel tree-based circuit representation is proposed. Its advantages over existing work include:

- It is a more compact representation that can be more efficiently mapped to circuit netlists, thanks to its “Device-circuit position" form of the tree.
- It lends itself to the adoption of crossover operators, whose behaviour corresponds to that of exchanging well defined sub-circuits between individuals. This is mainly an advantage over graph-based representation [17], but, together with the other evolutionary operators, it also helps the GP to achieve better fitness and Hits than existing work, including existing work based on Koza’s tree representation [11].
- It supports three-port devices, which is an advantage over previous work with topology-restricted representation [12].
- It prevents the problem of infeasible chromosomes by adopting three types of structure check, which is an an advantage over the fitness penalized-based method [13].

Manipulating memristors is very difficult in practice as the components are frequently unstable on many levels, such as static variation and retention problem. Our work made the primary attempt to apply an ideal HP memristor model and MOSFET to construct a memristor-based pulse generation circuit achieving the corresponding basic function. The feasibility of our evolved circuits is verified by the circuit simulations, indicating that they are feasible to be implemented with real physical memristors. For potential future work, the proposed approach can be adopted for evolving circuits with more realistic memristor models. Moreover, the scalability of our approach could also be improved further. We will investigate new approaches that can evolve much larger and more complex circuits. The evolved circuits could also be implemented using real physical hardware, to further validate them.

Appendix

The average fitness for 20 runs of the experiments (voltage reference circuit, temperature sensor circuit, Gaussian function generator and memristor-based pulse generation circuit) are shown in Fig. 24. As we can see from the figure, the fitness curve presents the typical fitness improvement behaviour of evolutionary algorithms, where the blue shadow indicates the standard error.

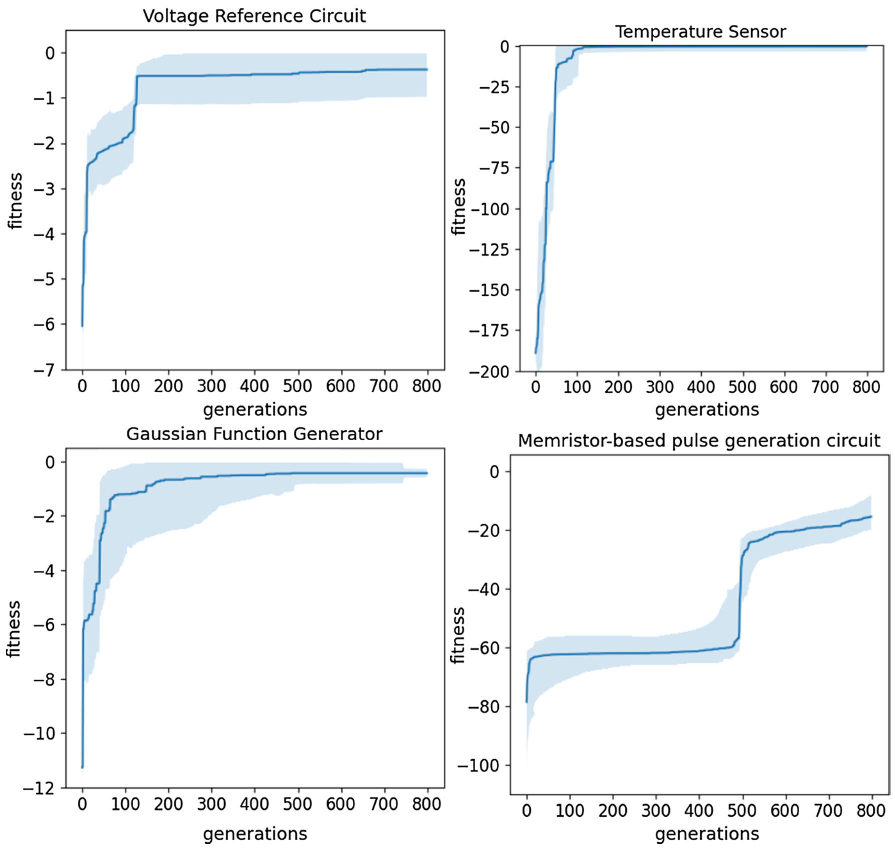


Fig. 24 The average fitness for 20 runs of the experiment on different tasks

Acknowledgements This work was supported by Guangdong Provincial Key Laboratory (Grant No. 2020B121201001), the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (Grant No. 2017ZT07X386), Shenzhen Science and Technology Program (Grant No. KQTD2016112514355531), and the Research Institute of Trustworthy Autonomous Systems (RITAS).

References

1. B. Liu, Y. Wang, Z. Yu, L. Liu, M. Li, Z. Wang, J. Lu, F.V. Fernández, Analog circuit optimization system based on hybrid evolutionary algorithms. Integration **42**(2), 137–148 (2009)
2. O. Mitea, M. Meissner, L. Hedrich, P. Jores, Automated constraint-driven topology synthesis for analog circuits. in *Proc. DATE 2001* (Grenoble, 2011), , pp. 1–4
3. A. Das, R. Vemuri, An automated passive analog circuit synthesis framework using genetic algorithms. in *Proc. IEEE ISVLSI 2007*, (Porto Alegre, 2007), pp. 145–152
4. E.S. Ochotta, R.A. Rutenbar, L.R. Carley, Synthesis of high-performance analog circuits in astrx/oblx. IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. **15**(3), 273–294 (1996)
5. X. Yao, T. Higuchi, Promises and challenges of evolvable hardware. IEEE Trans. Syst. Man Cybern. C **29**(1), 87–97 (1999)
6. W. Kruiskamp, D. Leenaerts, Darwin: CMOS OPAMP synthesis by means of a genetic algorithm. in *Proc. 32nd DAC 1995*, (San Francisco, 1995), pp. 433–438
7. M. O'Neill, C. Ryan, Grammatical evolution. IEEE Trans. Evol. Comput. **5**(4), 349–358 (2001)
8. J.R. Koza, D. Andre, F.H. Bennett III, M.A. Keane, Use of automatically defined functions and architecture-altering operations in automated circuit synthesis with genetic programming. in *Proc. 1st Annual Conference on Genetic Programming*, pp. 132–140 (Stanford, 1996)
9. Ž Rojec, Á. Bűrmen, I. Fajfar, Analog circuit topology synthesis by means of evolutionary computation. Eng. Appl. Artif. Intell. **80**, 48–65 (2019)
10. J.D. Lohn, S.P. Colombano, A circuit representation technique for automated circuit design. IEEE Trans. Evol. Comput. **3**(3), 205–219 (1999)
11. J.R. Koza, F.H. Bennett, D. Andre, M.A. Keane, F. Dunlap, Automated synthesis of analog electrical circuits by means of genetic programming. IEEE Trans. Evol. Comput. **1**(2), 109–128 (1997)
12. S.J. Chang, H.S. Hou, Y.K. Su, Automated passive filter synthesis using a novel tree representation and genetic programming. IEEE Trans. Evol. Comput. **10**(1), 93–100 (2006)
13. F. Castejón, E.J. Carmona, Automatic design of analog electronic circuits using grammatical evolution. Appl. Soft Comput. **62**, 1003–1018 (2018)
14. C. Mattiussi, D. Floreano, Analog genetic encoding for the evolution of circuits and networks. IEEE Trans. Evol. Comput. **11**(5), 596–607 (2007)
15. A. Das, R. Vemuri, A graph grammar based approach to automated multi-objective analog circuit design. in *Proc. DATE 2009*, pp. 700–705 (Nice, 2009)
16. J. He, M. Liu, Y. Chen, A novel real-coded scheme for evolutionary analog circuit synthesis. in *Proc. ISA 2009*, pp. 1–4 (Wuhan, 2009)
17. Z. Gan, Z. Yang, T. Shang, T. Yu, M. Jiang, Automated synthesis of passive analog filters using graph representation. Expert Syst. Appl. **37**(3), 1887–1898 (2010)
18. A. Mesquita, F.A. Salazar, P.P. Canazio, Chromosome representation through adjacency matrix in evolutionary circuits synthesis. in *Proc. the NASA/DoD Conference on Evolvable Hardware*, pp. 102–109 (2002)

19. J.R. Koza, D. Andre, M.A. Keane, F.H. Bennett III., *Genetic programming III: Darwinian invention and problem solving*, vol. 3 (Morgan Kaufmann, 1999)
20. O. Krestinskaya, A.P. James, L.O. Chua, Neuromemristive circuits for edge computing: a review. IEEE Trans. Neural Netw. Learn. Syst. (2019). https://doi.org/10.1109/TNNLS.2019.2899262
21. C.D. Schuman, T.E. Potok, R.M. Patton, J.D. Birdwell, M.E. Dean, G.S. Rose, J.S. Plank, A survey of neuromorphic computing and neural networks in hardware. arXiv:1705.06963 (2017)
22. X. Shi, Z. Zeng, L. Yang, Y. Huang, Memristor-based circuit design for neuron with homeostatic plasticity. IEEE Trans. Emerg. Top. Comput. Intell. **2**(5), 359–370 (2018)
23. L. Zhao, Q. Hong, X. Wang, Novel designs of spiking neuron circuit and STDP learning circuit based on memristor. Neurocomputing **314**, 207–214 (2018)
24. Z. Wang, X. Wang, A novel memristor-based circuit implementation of full-function Pavlov associative memory accorded with biological feature. IEEE Trans. Circ. Syst. I **65**(7), 2210–2220 (2017)
25. A. Sinha, M.S. Kulkarni, C. Teuscher, Evolving nanoscale associative memories with memristors. in *Proc. IEEE NANO 2001*, pp. 860–864 (Portland, 2011)
26. M.S. Kulkarni, C. Teuscher, Memristor-based reservoir computing. in *Proc. IEEE NANOARCH 2012*, pp. 226–232 (Amsterdam, 2012)
27. R. Gharpinde, P.L. Thangkhiew, K. Datta, I. Sengupta, A scalable in-memory logic synthesis approach using memristor crossbar. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **26**(2), 355–366 (2017)
28. H.P. Wang, C.C. Lin, C.C. Wu, Y.C. Chen, C.Y. Wang, On synthesizing memristor-based logic circuits with minimal operational pulses. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **26**(12), 2842–2852 (2018)
29. S.E. Sorkhabi, L. Zhang, Automated topology synthesis of analog and rf integrated circuits: A survey. Integration **56**, 128–138 (2017)
30. E.A. Klumperink, F. Bruccoleri, B. Nauta, Finding all elementary circuits exploiting transconductance. IEEE Trans. Circuits Syst. II **48**(11), 1039–1053 (2001)
31. J. He, J. Yin, Evolutionary design model of passive filter circuit for practical application. Genet. Program Evolvable Mach. **21**(4), 571–604 (2020)
32. J.B. Grimbleby, Automatic analogue network synthesis using genetic algorithms. in *Proc. GALESIA 1995*, pp. 53–58 (Sheffield, 1995)
33. A. Manazir, K. Raza, Recent developments in cartesian genetic programming and its variants. ACM Comput. Surv. **51**(6), 1–29 (2019)
34. J.R. Woodward, Ga or gp? that is not the question. In: The 2003 Congress on Evolutionary Computation, 2003. CEC'03., vol. 2, pp. 1056–1063. IEEE (2003)
35. J.R. Koza, Survey of genetic algorithms and genetic programming. In: Wescon conference record, pp. 589–594. WESTERN PERIODICALS COMPANY (1995)
36. T. Sripramong, The invention of cmos amplifiers using genetic programming and current-flow analysis. IEEE Trans. Comput Aided Des. Integr. Circ. Syst. **21**(11), 1237–1252 (2002)
37. W. Banzhaf, P. Nordin, R.E. Keller, F.D. Francone, *Genetic Programming* (Springer, New York, 1998)
38. J.F. Miller, S.L. Smith, Redundancy and computational efficiency in cartesian genetic programming. IEEE Trans. Evol. Comput. **10**(2), 167–174 (2006)
39. D. Chen, T. Aoki, N. Homma, T. Terasaki, T. Higuchi, Graph-based evolutionary design of arithmetic circuits. IEEE Trans. Evol. Comput. **6**(1), 86–100 (2002)
40. T. Aoki, N. Homma, T. Higuchi, Evolutionary synthesis of arithmetic circuit structures. Artif. Intell. Rev. **20**(3–4), 199–232 (2003)
41. M. Natsui, N. Homma, T. Aoki, T. Higuchi, Topology-oriented design of analog circuits based on evolutionary graph generation. in*Proc. PPSN*, pp. 342–351 (Birmingham, 2004)
42. J.A. Walker, J.A. Hilder, A.M. Tyrrell, Evolving variability-tolerant cmos designs. in *International Conference on Evolvable Systems*, pp. 308–319. Springer (2008)
43. J.R. Koza et al., *Genetic Programming II*, vol. 17 (MIT Press, Cambridge, 1994)
44. F. Wang, Y. Li, L. Li, K. Li, Automated analog circuit design using two-layer genetic programming. Appl. Math. Comput. **185**(2), 1087–1097 (2007)
45. A.C. Torrezan, J.P. Strachan, G. Medeiros-Ribeiro, R.S. Williams, Sub-nanosecond switching of a tantalum oxide memristor. Nanotechnology **22**(48), 485203 (2011)
46. I. Gupta, A. Serb, A. Khiat, R. Zeitler, S. Vassanelli, T. Prodromakis, Real-time encoding and compression of neuronal spikes by metal-oxide memristors. Nat. Commun. **7**(1), 1–9 (2016)

47. H. Kim, M.P. Sah, C. Yang, T. Roska, L.O. Chua, Memristor bridge synapses. Proc. IEEE **100**(6), 2061–2070 (2011)
48. S. Silva, E. Costa, Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. Genet. Program. Evol. Mach. **10**(2), 141–179 (2009)
49. M. O'Neill, R. Poli, W.B. Langdon, F. Nicholas, mcphee: a field guide to genetic programming (2009)
50. H. Vogt, M. Hendrix, P. Nenzi, Ngspice user's manual version 31 (describes ngspice release version) (2019)
51. L. Nagel, D.O. Pederson, Spice (simulation program with integrated circuit emphasis) (1973)
52. A.E. Eiben, J.E. Smith et al., *Introduction to Evolutionary Computing*, vol. 53 (Springer, Cham, 2003)
53. A.P. Brokaw, A simple three-terminal IC bandgap reference. IEEE J. Solid-State Circ. **9**(6), 388–393 (1974)
54. G.C. Meijer, Thermal sensors based on transistors. Sens. Actuators **10**(1–2), 103–125 (1986)
55. C. Popa, Low-voltage improved accuracy gaussian function generator with fourth-order approximation. Microelectron. J. **43**(8), 515–520 (2012)
56. D.B. Strukov, G.S. Snider, D.R. Stewart, R.S. Williams, The missing memristor found. Nature **453**(7191), 80–83 (2008)
57. A. Bofil l-iPetit, A.F. Murray, Synchrony detection and amplification by silicon neurons with STDP synapses. IEEE Trans. Neural Netw. **15**(5), 1296–1304 (2004)
58. G. Indiveri, A low-power adaptive integrate-and-fire neuron circuit. in *Proc. IEEE ISCAS 2003.*, vol. 4, pp. IV–IV (Bangkok, 2003)
59. J.M. Cruz-Albrecht, M.W. Yung, N. Srinivasa, Energy-efficient neuron, synapse and STDP integrated circuits. IEEE Trans. Biomed. Circ. Syst. **6**(3), 246–256 (2012)
60. F.T. Zohora, S. Debnath, A.H.U. Rashid, Memristor-cmos hybrid implementation of leaky integrate and fire neuron model. in *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, pp. 1–5. IEEE (2019)
61. I. Vourkas, A. Batsos, G.C. Sirakoulis, Spice modeling of nonlinear memristive behavior. Int. J. Circ. Theory Appl. **43**(5), 553–565 (2015)
62. D. Batas, H. Fiedler, A memristor spice implementation and a new approach for magnetic flux-controlled memristor modeling. IEEE Trans. Nanotechnol. **10**(2), 250–255 (2010)
63. B. Li, G. Shi, A native spice implementation of memristor models for simulation of neuromorphic analog signal processing circuits. ACM Trans. Design Autom. Electr. Syst. (TODAES) **27**(1), 1–24 (2021)
64. Y.N. Joglekar, S.J. Wolf, The elusive memristor: properties of basic electrical circuits. Eur. J. Phys. **30**(4), 661 (2009)
65. J. Zheng, Z. Zeng, Y. Zhu, Memristor-based nonvolatile synchronous flip-flop circuits. in *2017 seventh international conference on information science and technology (ICIST)*, pp. 504–508. IEEE (2017)
66. Z. Wang, X. Wang, Z. Lu, W. Wu, Z. Zeng, The design of memristive circuit for affective multi-associative learning. IEEE Trans. Biomed. Circuits Syst. **14**(2), 173–185 (2020)
67. M.R. Azghadi, B. Linares-Barranco, D. Abbott, P.H. Leong, A hybrid cmos-memristor neuromorphic synapse. IEEE Trans. Biomed. Circuits Syst. **11**(2), 434–445 (2016)
68. C. Wu, T.W. Kim, H.Y. Choi, D.B. Strukov, J.J. Yang, Flexible three-dimensional artificial synapse networks with correlated learning and trainable memory capability. Nat. Commun. **8**(1), 1–9 (2017)
69. V. Keshmiri, A study of the memristor models and applications (2014)
70. R. Williams, Finding the missing memristor. http://www.casttv. http://wn.com/Calit2ube (2010)
71. J.H. Wijekoon, P. Dudek, Compact silicon neuron circuit with spiking and bursting behaviour. Neural Netw. **21**(2–3), 524–534 (2008)

Authors and Affiliations

Xinming Shi[1,2] · **Leandro L. Minku**[2] · **Xin Yao**[1,2]

Xinming Shi
xxs972@cs.bham.ac.uk

Leandro L. Minku
l.l.minku@cs.bham.ac.uk

1 Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation, Department of Computer Science and Engineering, Southern University of Science and Technology (SUSTech), Shenzhen, China

2 School of Computer Science, University of Birmingham, Birmingham, UK