Received July 3, 2020, accepted July 18, 2020, date of publication July 24, 2020, date of current version August 5, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3011641

Introducing Modularity and Homology in Grammatical Evolution to Address the Analog Electronic Circuit Design Problem

FEDERICO CASTEJÓN AND ENRIQUE J. CARMONA

Departamento de Inteligencia Artificial, ETS Ingeniería Informática, Universidad Nacional de Educación a Distancia (UNED), 28040 Madrid, Spain

Corresponding author: Enrique J. Carmona (ecarmona@dia.uned.es)

ABSTRACT We present a new approach based on grammatical evolution (GE) aimed at addressing the analog electronic circuit design problem. In the new approach, called multi-grammatical evolution (MGE), a chromosome is a variable-length codon string that is divided into as many partitions as subproblems result from breaking down the original optimization problem: circuit topology and component sizing in our case. This leads to a modular approach where the solution of each subproblem is encoded and evolved in a partition of the chromosome. Additionally, each partition is decoded according to a specific grammar and the final solution to the original problem emerges as an aggregation result associated with the decoding process of the different partitions. Modularity facilitates the encoding and evolution of the solution in each subproblem. On the other way, homology helps to reduce the potentially destructive effect associated with standard crossover operators normally used in GE-based approaches. Seven analog circuit designs are addressed by an MGE-based method and the obtained results are compared to those obtained by different methods based on GE and other evolutionary paradigms. A simple parsimony mechanism was also introduced to ensure compliance with design specifications and reduce the number of components of the circuits obtained. We can conclude that our method obtains competitive results in the seven circuits analyzed.

INDEX TERMS Genetic programming, grammatical evolution, evolutionary electronics, analog circuit automatic design.

I. INTRODUCTION

Analog death has been predicted so many times, but the analog integrated circuit (IC) market growth rate is currently even greater (6.6%) than the average rate for the whole IC market (5.1%) [1]. A reason for this could be that the world is fundamentally analog [2] and so, some functions still have to remain analog, for example, transducers [2], RF communications [3], and low power applications [4].

Though there are powerful electronic design automation (EDA) tools which can almost fully automate the whole digital design process, this is not the case in analog design, where there is still no widely accepted tools [5]. Analog design is considered to be knowledge-intensive [4] and considerably more complex than digital design, even for small problem sizes [6]. Analog or mixed-signal designs typically need to optimize dozens of specifications, some of them conflicting, depending on the nonlinear behavior of the components [7]. Thus, a single change could affect the whole design since analog circuits are bidirectional at their boundaries [8]. Therefore, most of the constraints in today's analog design are still specified and considered manually by expert designers [6].

One of the alternatives for automatic design is evolutionary electronics (EE) [9]. The main goal of EE is to automatically synthesize electronic circuits by using evolutionary algorithms (EA). The works of Koza *et al.* (the late 90s), using genetic programming (GP), are outstanding in EE due to the broad set of evolved circuits [10]–[13]. Since then, many more approaches have emerged [9], [14]–[22]. However, there is some controversy when the reliability of circuits obtained by EE is considered. Reliability is an important property in the IC industry, where the worst fear is a respin, which happens when a fabricated IC does not work, and going back to the first stages of design is needed. In this way, a human designer may be somewhat skeptical about the unconventional designs produced by an EA. Conventional

The associate editor coordinating the review of this manuscript and approving it for publication was Gustavo Olague.

design can be constrained by the designer's own experience or geometrical preferences, like symmetry, which is not necessarily taken into account by an EA. The evolutionary process of an EA is just directed by the optimization of its fitness function and can find solutions out of human thinking limits [23]. No rules, nor expert knowledge are needed, that is, blocks are invented or reinvented from scratch and this gives its open-ended nature [24]. It has been said that the open-ended approach finds non-conventional solutions and provides both highly compact designs [19] and human-competitive designs which can be comparable to patented designs [25]. In another way, most human designers consider that EA designs can present strange topologies or without apparent logic [24]. Possibly, this is the most serious problem in the open-ended approach: getting from experimental designs to industrial designs [26].

Two are the main contributions of this work. First, we propose a new EA, called *multi-grammatical evolution* (MGE), oriented to solve the analog circuit design problem. Second, an MGE-based method is implemented and successfully applied to the automatic design of seven benchmark analog circuits. However, it is also necessary to mention the limitations of our study: (i) we only manage two levels of abstraction into the design process (topology and sizing); (ii) each circuit obtained by the proposed approach is expressed by its netlist, that is, a list of components that includes the connection nodes and parameter values of each component. Here, it should be noted that, from the perspective of industrial circuit design, getting a netlist is just the first step of a more complex process that implies several stages [27]: netlist, layout, photo-lithographic masks, wafer (chips), packaging, testing and debugging; (iii) the degree of compliance of each circuit with the design specifications is evaluated by simulation using NGSpice [28], an open-source electronic circuit simulator.

The rest of this manuscript is structured as follows: Section II shows different works related to our proposal. Section III describes MGE in detail. Then, Section IV presents several case studies related to the analog electronic circuit automatic design problem, which will be addressed by an MGE-based method. In Section V, the results obtained with our method are analyzed and compared with those obtained by a GE-based method and other evolutionary paradigms. Finally, the conclusions are presented in Section VI.

II. RELATED WORK

GP was introduced by John Koza in 1992 [29] and it has proved to be a very powerful algorithm in optimization problems. The Koza *et al.*'s works on GP covered a broad set of fields such as symbolic regression, control or circuit design [13], and since then, GP has achieved outstanding results in many other fields such as data mining [30], empirical modeling [31], convolutional neural network design [32], financial applications [33], and material strength prediction [34], among others. However, GP needs a special representation based on parse trees and, therefore, it requires special variation operators, different from those used to manage chromosomes represented by linear strings. Additionally, the variation operators in GP must guarantee the closure property, that is, the results of the variation operators must be valid chromosomes from a syntactic point of view [29]. In order to guarantee the closure property, the variation operators have to be carefully crafted for the specific problem and representation used.

To overcome the closure problem, several extensions of GP based on grammars appeared. They are called *grammar-based genetic programming* (GBGP) [35] or *grammar guided genetic programming* (GGGP) [36]. In particular, grammatical evolution (GE) is a GP variant based on grammar which was pioneered by Ryan *et al.* [37] and O'Neill and Ryan [38]. Unlike GP, GE uses linear strings as chromosomes and, consequently, can use standard variation operators. The closure property is also guaranteed by the decoding process based on a context-free grammar.

In GE, chromosomes are represented by variable-length binary strings, where the unit of information is called a *codon*, normally corresponding to one byte in the chromosome. The use of variable-length chromosomes is of special interest since it not only facilitates adjusting the solution to the necessary size but also favors the emergent property of self-organization, which is a natural step towards richer and more open-ended evolving system models [39]–[41]. The decoding of a chromosome consists of reading its codons from left to right. Each codon is used to choose an appropriate production rule in order to expand the current non-terminal symbol in the expression that is being expanded. The choice of the *rule* is made according to the following mapping function:

$$rule = codon\_value \text{ MOD } NR \tag{1}$$

where MOD is the modulus function, *codon_value* is the value of the codon and *NR* is the number of rules for the current non-terminal symbol. Additionally, if all the codons of a chromosome are completely read, but the expression is not fully expanded, then the reading restarts at the beginning of the chromosome and the decoding process continues. This mechanism is known as *wrapping* [38]. The wrapping parameter defines how many times the chromosome can be read before giving up. The decoding process just described is also called depth-first mapping. However, there are other possibilities of mapping in GE [42], such as breadth-first, random, or πGE [43].

According to a recent survey [44], *grammatical-GP* (which includes GE) was the second variant most used by practitioners in the field of GP, only preceded by *GP-standard*. However, there is some controversy about GE performance compared to other GGGPs [35], [45], [46]. From the beginning, GE was based on the use of a standard one-point crossover operator inspired by genetic algorithms (GA) [38]. Although it is a simple and valid operator, its main disadvantage is a potentially destructive behavior when it is used in GE. Alternatively, the idea of homologous crossover can be used,

which draws inspiration from biology [47], [48]. Homology in nature implies that the chromosome fragments to be exchanged always belong to the same position and are of similar size. However, there is evidence that the attempts made to build homologous operators in GE have not given better results than those obtained by the one-point operator [49]. Besides, a homologous operator is not easy to implement in GE because it requires storing the history of production rules used in the decoding of each chromosome.

Additionally, modularity and its benefits are well known in computer science [50] or other more specific areas such as graph theory or network analysis [51]–[53]. Modularity allows us to address a problem by decomposing it into subproblems or modules. In this way, each module can focus on just one aspect or functionality of the original problem, encapsulating the initial complexity in each module. Modularity is also a recurrent theme in biology because it facilitates evolvability by limiting interference between the adaptations of different functions [54].

MGE, the new approach proposed here, is based on GE, but includes two important properties: modularity and homology. The main idea of MGE is to decompose the original optimization problem into different subproblems to encode and evolve the solution to each subproblem in a different partition of the chromosome. The final phenotype is obtained by decoding each partition of the chromosome with a specific grammar. Each grammar allows the decoding process to build solutions belonging to the search space of each subproblem. In this context, we believe that the analog circuit design problem is ideal to test the potential of MGE and, therefore, to carry out a first analysis of the effects of simultaneously including the properties of homology and modularity in GE. In particular, the analog circuit design process can be easily decomposable into two subproblems: circuit topology selection and component sizing. Therefore, the idea is to define a grammar to decode the topology, another grammar to decode the sizing of the components that result from applying the first grammar, and finally, use the MGE formalism to coordinate the entire decoding process resulting of using both grammars.

GP and other of its variants, such as Gene Expressing Programming [55], also have the possibility of partitioning the chromosome. In this context, each partition can encode the potential solution of the different subproblems in which the main problem can be decomposed and, additionally, as it is shown in [56], this form of representation can be used by the variation operators at different levels (partition level or gene level). However, the representational power of the formal grammars used by GE is worth exploring further. As a grammar is used, it is trivial to modify the output solution structures by editing the grammar. It is also possible to inject domain knowledge in the grammar definition, allowing us to reduce the search space of all feasible solutions. Therefore, MGE is designed to incorporate the homology and modularity properties in GE, while maintaining the advantage of using the powerful idea of grammar.

III. METHOD DESCRIPTION

In this section, MGE is presented. First, the way to encode the information in the chromosome and the decoding process associated with this new approach are described. Second, given that the analog circuit design problem is directly decomposable into two subproblems (circuit topology and component sizing), and MGE need to define one grammar for each subproblem, we present a generic grammar for topology and another for sizing. Then, an MGE decoding example shows how to obtain the phenotype (circuit) from a codon string (genotype). Finally, we also detail how to introduce homology property in MGE and how to define a type of crossover operator that takes advantage of this property.

A. MULTI-GRAMMATICAL EVOLUTION

A chromosome in MGE is a variable-length codon string that is divided into as many partitions as subproblems result from breaking down the original optimization problem. Each partition is decoded according to a specific grammar and the final solution to the original problem emerges as an aggregation result associated with the decoding process of the different partitions of the chromosome. MGE can be seen as a top-down approach, where the first grammar decodes high-level features of abstraction, and the last grammar decodes low-level features of abstraction. This approach based on modularity is inspired by nature, where different parts of the chromosome regulate different biological functions. In particular, modularity in MGE allows each partition of the chromosome to evolve in order to solve each subproblem.

The decoding process of MGE is based on GE, but it is modified to use several grammars sequentially. The process begins using the start symbol of the first grammar and works in the same way as in GE: each codon of the chromosome is read, and the standard modulus rule described in (1) is applied to expand the start symbol of the first grammar. The difference from GE appears when a *fully expanded expression* is obtained in the scope of the first grammar, that is, the expression only depends on the terminal symbols of this grammar. At this point, the decoding process continues in the following codon to the last one read, but now the decoding process makes use of the next grammar. The second grammar is built to have a special start symbol, called *start symbol string*, which is replaced by the fully expanded expression obtained in the previous step. The result of the second decoding is a new fully expanded expression that only depends on the terminal symbols of the second grammar. Then, this process is repeated until the last grammar is used. The final fully expanded expression obtained corresponds to the phenotype associated with the entire chromosome and, therefore, to a potential solution to the original problem.

MGE determines that a partition point, called *P-point*, will be marked in the chromosome at the point where the decoding process finishes with each grammar. If the decoding process for the last grammar ends before traversing the entire chromosome, the codons placed to the right of the last P-point are

simply ignored. The set of P-points divides the chromosome into different partitions, considering that each of them was decoded using its respective grammar. Note that partitions only appear when the decoding process has finished and, therefore, they are specific of each chromosome. The length of each partition is not fixed and depends strictly on the chromosome codon values, the specific grammar being used, and the fully expanded expression used as a start symbol string.

A graphic example of the decoding process (using three grammars) and the resulting chromosome partitions are shown schematically in Fig. 1. It can be seen how the fully expanded expression obtained by decoding the i-th partition of the chromosome is used as a start symbol string by the $(i+1)$-th grammar to decode the $(i+1)$-th partition of the chromosome. In order to present the decoding process more formally, algorithm 1 describes it in pseudocode. A practical example of the decoding process will also be shown in Section III-C.

Algorithm 1 MGE Decoding Process

Inputs: chromosome to decode (*chromosome*); start symbol of the first grammar (*start_symbol*); grammars to be used (*grammar*[*i*], with $i = 1, \ldots, n$)
Output: decoded solution (*expression*)

list_of_P-points ← {Ø};
reading_pointer ← 0;
expression ← *start_symbol*;
FOR i ← 1 **TO** n
  WHILE there are non-terminal symbols regarding the grammar[i] in *expression* **DO**
    non_terminal ← **next_non-terminal_symbol**(*expression*, *grammar*[*i*]);
    IF **production_rule**(*non_terminal*, *grammar*[*i*]) has several rules **THEN**
      reading_pointer ← *reading_pointer*+1;
      codon ← **read_chromosome**(*reading_pointer*, *chromosome*);
      rule ← **mod**(*codon*, **number_of_choices**(*non_terminal*, *grammar*[*i*]));
    ELSE
      rule ← 0;
    End-IF
    expression ← **expand_production_rule**(*expression*, *rule*);
  End-WHILE
  list_of_P-points ← *list_of_P-points* ∪ *reading_pointer*;
End-FOR

MGE, as a variant of GE, is also an evolutionary algorithm that allows us to separate the decoding process from the search process. Therefore, different search engines can be used, such as a genetic algorithm [38], differential evolution [57], or particle swarm optimization [58], [59]. Likewise, as variation and selection operators, anyone that works with GE could also be used in MGE. However, as we describe below, it will be necessary to design a specific crossover operator if we want to take advantage of the homology property.

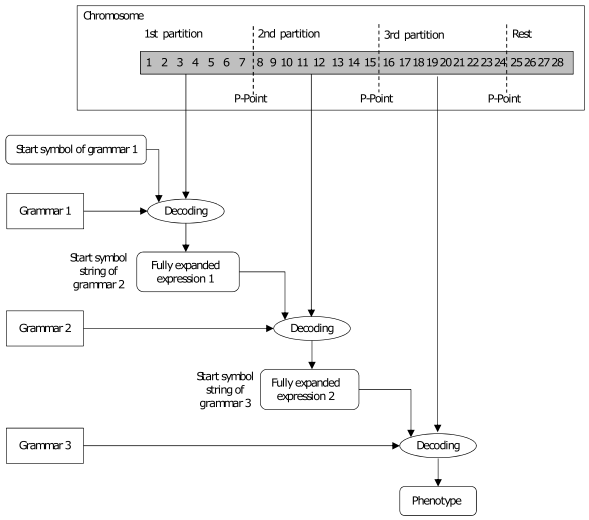


FIGURE 1. Example of the MGE-based decoding process for three grammars. The chromosome partitions are also shown.

In principle, a crossover operator could take advantage of the chromosome partitions obtained. This operator could focus on interchanging entire partitions of each parent using a P-point as a crossover point (XO-point), such as is shown in Fig. 2. However, those partitions inherited by each child and originally located in their parents to the right of each XO-point correspond to estimated partitions, since the actual partitions only appear after decoding. In other words, this kind of crossover operator does not guarantee the conservation of all the parents' partitions in their children and cannot, therefore, be considered a homologous operator. Anyhow, in Section III-D, we will describe a way to introduce the homology property in MGE.

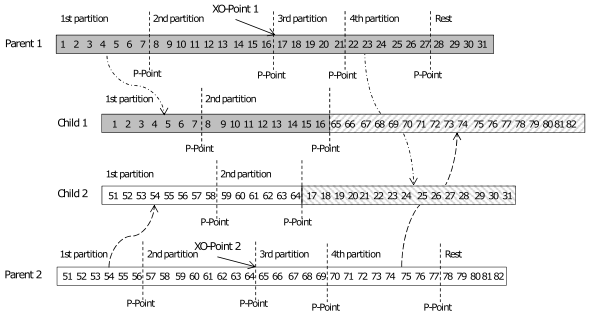


FIGURE 2. Example of a pseudo-homologous crossover operator in which the crossover points (XO-Points) must correspond to P-points in both parents. Note that only the parents' partitions located to the left of the XO-Points are preserved in their respective children. The 3rd and 4th partitions are not shown in the children because they will only appear after the decoding process.

Though grammars in GE are normally expressed in Backus-Naur form (BNF) notation, we use an *extended* BNF notation, which is an ISO standard [60]. Additionally, we pro-

pose here the use of grammar-based extensions. In general, grammar-based extensions are introduced to facilitate the exchange of knowledge between genotype and phenotype and to provide advantages in terms of convenience and potential improvement in performance [61]. Grammar-based extensions are also used to improve the expressiveness of grammars and obtain other functionalities. In this sense, some grammar-based extensions from [61] and new ones are used here. However, the grammars used are no longer compatible with ISO / IEC, nor are they pure BNF grammars.

Each grammar-based extension is enclosed in angle-brackets and is identified by a keyword. Some arguments may be needed depending on the type of extension used. The extensions proposed here are the following:

1) <GEPpointMarker> This extension allows us to set a P-point mark in the chromosome. The mark is applied at the codon that is decoded as a P-point by the current grammar. This kind of mark is used by crossover operators.
2) <GECodonValue: *start*, *end*> This extension is used as a convenient shorthand instead of an exhaustive enumeration of an integer interval. It tells the decoding process to draw a codon from the chromosome and expand it in the range of integers delimited by the *start* and *end* values.
3) <GEResult> This extension tells the algorithm to continue the decoding using the previous fully expanded expression. It is just a formalism to allow the fully expanded expression obtained by the (i)-th grammar to be used as a start symbol string by the $(i+1)$-th grammar.

B. GRAMMARS FOR ANALOG CIRCUIT DESIGN

In this section, the grammars proposed for analog circuit design are shown. Since there are two well-differentiated tasks for circuit design: topology selection and component sizing, each grammar focuses on one of these tasks. Both grammars, like those presented in [62], are oriented to directly generate circuit netlists. Grammars are defined with the tuple $< S, T, N, R, E >$, where S is the start symbol (topology grammar) or the start symbol string (sizing grammar), T is the set of terminal symbols, N is the set of non-terminal symbols, R is the set of production rules, and E is the set of grammar-based extensions.

1) TOPOLOGY GRAMMAR

The goal of the topology grammar (see table 1) is to determine the circuit topology as a netlist. The start symbol of this grammar is the non-terminal symbol LIST, which is directly expanded as the non-terminal symbol COMPONENTS. At the same time, COMPONENTS can be expanded as a circuit component, as a new non-terminal symbol COMPONENTS (which allows the netlist to be expanded) or as a non-terminal symbol END (which marks the ending of the netlist and introduces a P-point marker using the grammar extension <GEPpointMarker>). The definition of this grammar is generic, that is, depending on the design requirements of a particular circuit, some types of components shown in the grammar can be removed or new ones can be added by the user. For example, one of the two types of transistors (BJT or MOSFET) could be removed or a new type of component, for example, an INDUCTOR, could be added.

TABLE 1. Grammar for topology selection (see Section III-B for a detailed description).

S =	LIST
T =	{ "R", "C", "Q", "M", "null1", "null2", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "e", ".", "RESISTORVAL", "CAPACITORVAL", "BJTTYPE", "MOSTYPE", "CHANNELWIDTH", "ENDFIRST", end-of-line character}
N =	{ LIST, COMPONENTS, RESISTOR, CAPACITOR, BJT, MOSFET, DUMMY, NODE, END, EOL }
E =	{ <GEPpointMarker>, <GECodonValue> }
"R" *comprises the following rules of production:*	
LIST =	COMPONENTS;
COMPONENTS =	RESISTOR, END | RESISTOR, COMPONENTS | CAPACITOR, END | CAPACITOR, COMPONENTS | BJT, END | BJT, COMPONENTS | MOSFET, END | MOSFET, COMPONENTS;
RESISTOR=	"R", NODE, NODE, DUMMY, "RESISTORVAL", EOL;
CAPACITOR =	"C", NODE, NODE, DUMMY, "CAPACITORVAL", EOL;
BJT =	"Q", NODE, NODE, NODE, "BJTTYPE", EOL;
MOSFET =	"M", NODE, NODE, NODE, "MOSTYPE", "CHANELWIDTH", EOL;
NODE =	<GECodonValue: 0, MNN + test_fixture_nodes-1 >;
DUMMY =	"null1" | "null2";
END =	"* ENDFIRST", EOL, <GEPpointMarker>;
EOL	*end-of-line character*

Note how the grammar extension <GECodonValue> is used here to define the list of nodes that can be used in the circuit to evolve (see production rule for the non-terminal symbol NODE). The minimum number of nodes is given by the so-called *test fixture*, that is, the circuit part that does not evolve (inputs, outputs, ground, power supply sources, etc.). The test fixture is defined by the design specifications and contributes with a fixed number of nodes. The total number of nodes in the circuit to evolve is given by the sum of nodes used by the test fixture and the maximum node number (MNN), which is established by the user.

The topology grammar is defined to always use fixed-size blocks (4 codons). This type of grammar is known as *block-grammars* [62]. To meet that restriction in those types of components that would need less than four codons to be decoded, a non-terminal symbol, called DUMMY, is introduced. The idea is to force the decoding process to consume as many codons as necessary until completing the sequence of 4 codons. This is implemented in the grammar by making a production rule for the DUMMY symbol with two options: ''null1'' and ''null2''. Note that when a production rule has only one option associated, no codon is read [38]. Therefore, the existence of two options for the DUMMY symbol forces a codon to be read when this non-terminal symbol is being expanded.

2) SIZING GRAMMAR

The goal of the sizing grammar (see table 2) is to continue the decoding process from the partial netlist decoded by the topology grammar and, therefore, determine the value of each component. For this, in the expression resulting from applying the topology grammar, those symbols that are non-terminal, regarding the sizing grammar, will have to be expanded. For example, RESISTORVAL or CAPACITORVAL are terminal symbols for the topology grammar, but they are non-terminal in the sizing grammar. As is shown in algorithm 1, when there is a change of grammar, the decoded expression by the i-th grammar is considered the start symbol of the $(i+1)$-th grammar. Therefore, that expression is traversed from left to right and every symbol is checked to see if it is non-terminal in the new grammar. If so, the symbol is expanded.

TABLE 2. Grammar for component sizing (see Section III-B for a detailed description).

S =	LIST
T =	{ "R", "C", "Q", "M", "QNPN", "QPNP", "NMOS1 L=10u W=", "PMOS1 L=10u W=", "null1", "null2", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "e", ".", "END", end-of-line character}
N =	{ LIST, BJTTYPE, TYPE, MOSTYPE, MODELNMOS, MODELPMOS, DUMMY, NODE, RESISTORVAL, CAPACITORVAL, CHANELWIDTH, DIGIT, NONZERODIGIT, EXPONENT, ENDFIRST, EOL }
E =	{ <GEResult>, <GEPpointMarker>, <GECodonValue> }
"R" *comprises the following rules of production:*	
LIST =	<GEResult>;
RESISTORVAL =	NONZERODIGIT, ".", DIGIT, "e", DIGIT, DUMMY;
CAPACITORVAL =	NONZERODIGIT, ".", DIGIT, "e", EXPONENT, DUMMY;
BJTTYPE =	TYPE, DUMMY, DUMMY, DUMMY;
TYPE =	"QNPN" | "QPNP";
MOSTYPE =	MODELNMOS | MODELPMOS;
MODELNMOS =	"0", "NMOS1 L=10u W="
MODELPMOS =	"1", "PMOS1 L=10u W="
CHANELWIDTH =	NONZERODIGIT, DIGIT, "u" | "1", DIGIT, DIGIT, "u";
DIGIT =	<GECodonValue: 0, 9>;
NONZERODIGIT =	<GECodonValue: 1, 9>;
EXPONENT =	<GECodonValue: -12, -3>;
DUMMY =	"null1" | "null2";
ENDFIRST =	"END", <GEPpointMarker>;
EOL	*end-of-line character*

The start symbol of the sizing grammar is the non-terminal symbol LIST, which is expanded with the grammar extension <GEResult>. This grammar extension tells the algorithm to expand LIST using the fully expanded expression obtained by the topology grammar.

The non-terminal symbol ENDFIRST allows us the introduction of a P-point marker through the use of the grammar extension <GEPpointMarker>. Another grammar extension used is <GECodonValue> (see the production rule for the non-terminal symbols DIGIT, NONZERODIGIT, and EXPONENT).

RESISTORs and CAPACITORs have a value expressed in scientific notation, $m \cdot 10^e$, where $1 \leq m < 10$ and the magnitude order range of the exponent, e, depends on the type of component (resistor or capacitor). TRANSISTORs can be BJT or MOSFET. Two types of BJT and MOSFET transistors are defined by the non-terminal symbols BJTTYPE and MOSTYPE, respectively. Note that this grammar is also defined to utilize 4-codon size blocks (block-grammar) by using the non-terminal symbol DUMMY when necessary.

Finally, it should be noted that some chromosomes can be inexpressible, that is, chromosomes that cannot be decoded (the wrapping threshold is exceeded) or can produce circuits that cannot be simulated (unfeasible circuits). The fitness of this type of individual is strongly penalized in order to reduce the probability that they are selected as parents [62].

C. DECODING EXAMPLE

In this section, an example of how to use the topology and sizing grammars to decode a chromosome is shown. The example chromosome to be decoded is: (5, 21, 42, 14, 1, 34, 10, 7, 2, 94, 23, 8, 0, 12, 38, 15, 0, 20, 53, 78, 0, 100, 83, 111, 76, 29). Note that this chromosome can represent an individual from the initial population, where the codon values are randomly obtained, or an individual from the population in the i-th generation, where the codon values are the result of the evolutionary process. We assume in this example that the test fixture comprises four accessible nodes (see Fig. 3) and the MNN=6, so the production rule for the non-terminal symbol NODE will have ten options {0,1,..., 9}. A node is called *accessible* if it can be used by the evolved circuit. In other case, it is denominated *inaccessible* (e.g. the node linking Rs and Vs in Fig. 3).

The first stage of the decoding process, as indicated in Section III-A and using the topology grammar (see table 1), is as follows:

1) The decoding starts with the start symbol S, which is expanded as the non-terminal symbol LIST. The new expanded expression is: **LIST**.
2) Since the production rule for LIST has only one option, no codon is drawn and the new expanded expression is: **COMPONENTS**.
3) A codon value is needed for the expansion of COMPONENTS and the value of the first codon is 5, so $5\,MOD\,8 = 5$ selects rule *#5,* which produces the following expanded expression: **BJT, COMPONENTS**.
4) The production rule for BJT has only one option. Therefore, it is expanded directly without consuming codons. The new expanded expression is: **"Q", NODE, NODE, NODE, "BJTTYPE", EOL, COMPONENTS**.
5) "Q" is already a terminal symbol.
6) The non-terminal symbol NODE has to be expanded, so a new codon is read (value 21), and the modulus operator generates an integer in the range $(0, 9)$ by using the modulus operator $(21\,MOD\,10 = 1)$. Then, the node number 1 is chosen and the new expanded

expression is: **"Q" 1, NODE, NODE, "BJTTYPE", EOL, COMPONENTS**.

7) The expansion of the other two non-terminal symbols NODE involves reading the codons 42 and 14. The modulus operator produces node numbers 2 and 4, respectively, and the new expanded expression is: **"Q" 1 2 4, "BJTTYPE", EOL, COMPONENTS**.
8) "BJTTYPE" is a terminal symbol (no expansion is required) and the production rule for EOL has only one option, which is expanded as an end-of-line character. Therefore, from the perspective of our topology grammar, the first component of the netlist is decoded completely.
9) The non-terminal symbol COMPONENTS is expanded with codon value 1, selecting rule #1, which produces the following expanded expression: **[...] RESISTOR, COMPONENTS**. The symbol [...] represents the first component of the netlist, which is no longer displayed in the current expanded expression.
10) The production rule for RESISTOR has only one option and, therefore, it is expanded without consuming codon. The expanded expression is now: **[...] "R", NODE, NODE, DUMMY, "RESISTORVAL", EOL, COMPONENTS**.
11) The decoding process would continue in this way until all the symbols in the current expanded expression were non-terminals symbols regarding the topology grammar. In particular, the first twelve codons are used, a P-point is inserted after the twelfth codon, and the following fully expanded expression is obtained:

Q 1 2 4 BJTTYPE
R 4 0 null2 RESISTORVAL
C 4 3 null1 CAPACITORVAL
* ENDFIRST

Note that only the components and their connection nodes are specified (topology), but not the values that parameterize each component (sizing). Fig. 3 shows the circuit associated with the evolved netlist example and how it is connected to its test fixture. On the one hand, the test fixture is represented by the circuit components that do not evolve (the ground connection, the power supply, the signal source and its serial resistor, and the load resistor) and their associated nodes (0, 1, 2 and 3, respectively). On the other hand, the evolved circuit correspond to the evolved netlist: transistor (Q), capacitor (C), resistor (R), and their connections.

However, the above expression cannot be considered the final phenotype because there are still non-terminal symbols regarding the sizing grammar. Therefore, using the partial netlist obtained as a start symbol string and the sizing grammar as the current grammar, the decoding process continues as follows:

1) The <GEResult> grammar expansion allows us to continue the decoding process with the partial netlist obtained as a start symbol string.
2) The first non-terminal symbol, considering the sizing grammar, is BJTTYPE. The production rule for this symbol has only one option. Therefore, the new expanded expression is now: **Q** 1 2 4 **TYPE, DUMMY, DUMMY, DUMMY [...]**. The symbol [...] refers to the rest of the netlist obtained by the topology grammar.
3) The production rule for the non-terminal symbol TYPE has two options. The following codon is read (value 0) and the modulus operator selects the option QNPN. The new expanded expression is: **Q** 1 2 4 **QNPN, DUMMY, DUMMY, DUMMY, [...]**.
4) The following three non-terminal symbols DUMMY are expanded with the codon values 12, 38, and 15, producing null1, null1, and null2, respectively. The new expanded expression is now: **Q** 1 2 4 **QNPN null1 null1 null2 [...]**.
5) The next non-terminal symbol appears in the second line of the topology netlist (R 4 0 null2 RESISTORVAL) and corresponds to RESISTORVAL. The production rule of this symbol has only one option, and therefore, the new expanded expression is: **[...] R** 4 0 **null2, NONZERODIGIT, ". ", DIGIT, "e", DIGIT, DUMMY [...]**.
6) The non-terminal symbols NONZERODIGIT, DIGIT, and DIGIT are expanded with the grammar expansion <GECodonValue>. Their expansion involves the use of codons 0, 20, and 53, respectively, producing the digits 1, 0 and 3, respectively. The new expanded expression is: **[...] R** 4 0 **null2** 1.0e3**, DUMMY [...]**.
7) The expansion of the non-terminal symbol DUMMY with the codon 78 produces a null1, so the new expanded expression is: **[...] R** 4 0 **null2** 1.0e3 **null1 [...]**.
8) The decoding process would continue in this way until all the symbols in the current expanded expression were non-terminals symbols regarding the sizing grammar. In particular, the fully expanded netlist is as follows:

Q 1 2 4 QNPN null1 null1 null2
R 4 0 null2 1.0e3 null1
C 4 3 null1 1.0$e-9$ null2
* END

Note that the last codon read was in position 24 (value 1). Therefore, the codons located in positions 25 and 26

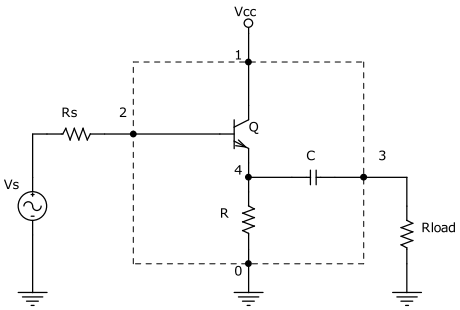


FIGURE 3. Example of the topology of an evolved circuit (inside the dashed line box). The test fixture corresponds to the set of components placed outside the box.

(i.e. values 76 and 29) were ignored in this case. Besides, the decoding process inserts a P-point at the last codon read. However, the process does not end here, since the evolved netlist obtained must be post-processed before being simulated. This stage includes, among other steps (see [62]), the incorporation of the test fixture netlist (see Fig. 3). The final simulatable netlist is the following:

**Test fixture netlist*
Vcc 1 0 dc 5
Vs 0 50 0.0 ac 0.001
Rs 50 2 1k
Rload 3 0 1k
**Evolved netlist*
Q 1 2 4 QNPN
R 4 0 1k
C 4 3 1n
* END

D. INTRODUCING HOMOLOGY IN MGE

As seen in Section III-A, the use of partitions apparently induces an interchromosomal structural homology, but such homology is only fictitious, given that some of the parents' partitions inherited by the children are altered in their decoding process. Therefore, a crossover operator based on the simple exchange of partitions does not guarantee the homology property. In this section, we present a way to introduce this property when MGE is applied to hierarchically decomposable design problems in general, and the analog electronic circuit design problem in particular.

The goal of most design problems is to learn a model that accomplishes a set of specifications and can be characterized by a hierarchy of n abstraction levels. Level 1 contains the decomposition of the original model into different components; level 2 contains the decomposition into subcomponents of each component belonging to level 1; and so on, for the different levels of abstraction from 3 to $n-1$. Finally, each component of level n stores the parameter values that characterize each component belonging to level $n-1$. Figure 4 shows an example where the original problem is divided into three levels of abstraction. For example, the first level could represent different stages of an amplifier. Then, the second level would represent the electronic components (and their connections) associated with each amplifier stage, and the third level would represent the value of each electronic component used in the previous level. However, if we want to introduce the homology property when MGE is applied to this kind of problems, it will be necessary to impose that each grammar only supports recursiveness on the right in the rule that defines the components to be utilized, that is:

$$C = C_1, C|\ldots|C_n, C|C_1|\ldots|C_n \tag{2}$$

where $C_1, \ldots, C_n$ are non-terminal symbols that represent the different components that can be used in a level of abstraction. In contrast, the following types of recursion would not be supported:

$$C = C, C_1|\ldots|C, C_n|C_1|\ldots|C_n \tag{3}$$
$$C = C, C_1, C|\ldots|C, C_n, C|C_1|\ldots|C_n \tag{4}$$

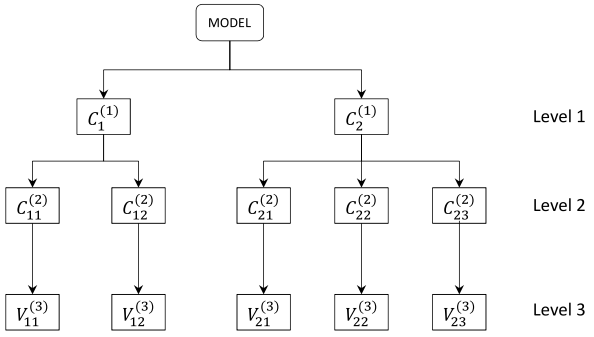


FIGURE 4. Example of a design model described by a hierarchy of three abstraction levels.

The motivation for the imposed constraint is to use blocks of consecutive codons in the chromosome when each component is decoded by using a grammar (block-grammar). In this way, each partition is divided into a set of blocks, with each block representing a component. Besides, a relationship can be established between each component and its corresponding decomposition (one or more components) in the level immediately below, that is, there is a one-to-one relationship between each block of the i-th partition and a subset of consecutive blocks of the $(i+1)$-th partition (see Fig. 5). Therefore, two types of homologies emerge: inter- and intra-chromosomal homology. The former establishes that the XO-Points must be chosen at the beginning of a block in the first partition of each parent. The latter determines that the blocks exchanged in the respective children must take into account the relationship between the respective blocks (or set of blocks) of consecutive partitions in each parent.

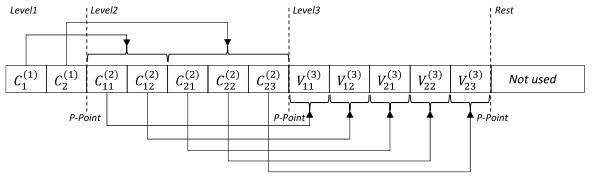


FIGURE 5. Chromosome that encodes an instance of the design model presented in Fig. 4. The one-to-one relationship between each component (block) of the i-th level (partition) and its corresponding decomposition (subset of consecutive blocks) in the $(i+1)$-th level (partition) allows us to define an intrachromosomal homology.

To illustrate an example of a crossover operator that takes advantage of the two types of homologies defined above, we return to the analog electronic circuit design problem. We must bear in mind that this problem is already a hierarchically decomposable problem, where two levels of abstraction are used. Concerning the grammars used (see tables 1 and 2), both of them meet the restriction of being recursive on the right and, in addition, are block-grammars, generating

4-codon fixed-size blocks. Thus, after the decoding process is finished, each electronic component is encoded into two intra-homologous blocks: one block in the topology partition and one block in the sizing partition. Therefore, if the topology partition encodes a number N of components, then N blocks will compose this partition and, since each block of the topology partition has a corresponding block in the sizing partition, there will also be N blocks in the sizing partition. Additionally, the related blocks appear in the same order in both partitions. Fig. 6 shows an example of the chromosome structure described here. Note that, if the decoding ends before consuming all codons, there would be a third partition comprising the ignored codons.

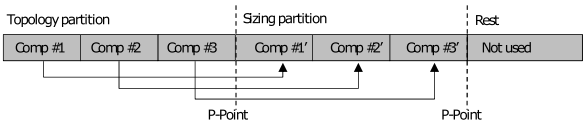


FIGURE 6. Example of chromosome partition for the circuit design problem. A partition for each grammar is used. Different blocks appear in each partition. Note that each block in the topology partition has its intra-homologous block in the sizing partition.

In this context, we have defined a new crossover operator, called *block-grammar-based multipart homologous crossover* (BG-MHX), that takes advantage of the information associated with the resulting chromosomal structure: (i) the netlist information is divided into the two chromosomal partitions; and (ii) each partition is formed by blocks, where each block in one partition has its intra-homologous block in the other. The BG-MHX operator acts as follows: First, a block is randomly selected in the topology partition of each parent, and the first codon of the selected block is marked as an XO-point. Then, the intra-homologous block is automatically determined in the sizing partition of each parent. Therefore, the components chosen to be copied from the first partition of one parent determine the corresponding blocks of the second partition be copied too. Thus, the complete information of each component (component type, connection nodes, and parameter values) is copied in the respective child. In this way, we avoid the destructive behavior associated with a classic crossover operator that could exchange partial information of two different components. Figure 7 shows an example of using the BG-MHX operator. Note that, usually, both parents are of different length, since MGE uses variable-length chromosomes. The BG-MHX operator definition also includes a maximum chromosome length constraint for the children obtained. If this constraint is violated, the child is clipped to the maximum length allowed. This constraint helps to mitigate the well-known bloat effect. Additionally, if wrapping took place when decoding any of the two parents, the crossover operator just clones both of them.

IV. CASE STUDIES

In this section, the specifications of seven benchmark circuits are presented. Additionally, the parameter configuration of the MGE-based implemented method is also detailed.

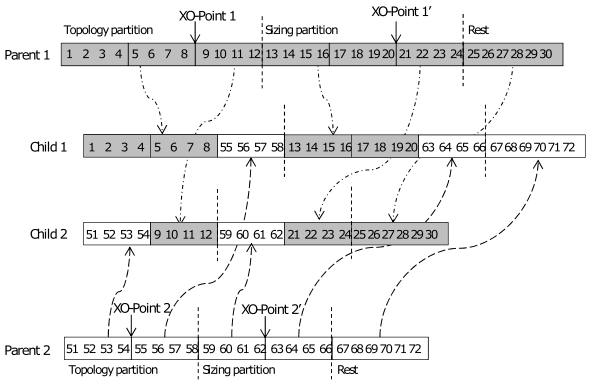


FIGURE 7. Example of using the BG-MHX operator for the circuit design problem. Note how the choice of the XO-Points 1 and 2 determines the choice of XO-Points 1' and 2' (intra-homologous blocks), respectively.

A. PROBLEM SPECIFICATIONS

The goal in the analog circuit design problem is to obtain a circuit that meets a set of requirements. The circuits tested here are the same as in [62], where two sets of benchmark circuits were used. The first set, denoted by *non-computing circuits*, comprises a temperature sensor circuit, a voltage reference circuit, and a Gaussian function generator circuit. A second set, denoted by *computational circuits*, comprises four computing circuits: squaring, square root, cubing, and cube root computing circuits.

Every circuit is evaluated at a number of fitting points. The fitness function is defined as the weighted absolute sum of the differences between the expected output voltage, V_{o_i}, and the measured output voltage, $\widetilde{V}_{o_i}$, at each of the fitting points. If $|V_{o_i} - \widetilde{V}_{o_i}|$ is lower than or equal to a given threshold value V_{th}, a hit is scored and, if all hits are scored, the circuit is considered a success. Equations (5) and (6) show the fitness function and the weights applied, respectively, in all the experiments. The circuit specifications are described briefly below (see [62] for a more detailed description).

$$fitness = \sum_i w_i |V_{o_i} - \widetilde{V}_{o_i}| \tag{5}$$

$$w_i = \begin{cases} 1.0 & \text{if } |V_{o_i} - \widetilde{V}_{o_i}| \leq V_{th} \\ 10.0 & \text{otherwise} \end{cases} \tag{6}$$

1) TEMPERATURE SENSOR CIRCUIT

This circuit takes its own temperature as input and must provide an output voltage proportional to the temperature value. The range of temperature is 0 °C $\leq T \leq$100 °C and the output voltage should be proportional to the temperature and normalized in the interval 0 V $\leq V_o \leq$ 10 V. Candidate circuits will be simulated with NGSpice using a voltage DC sweep analysis in the mentioned interval with a step size of 5 °C, giving a total of 21 fitting points. The threshold used here to consider a success is $V_{th} = 0.1$ V.

2) VOLTAGE REFERENCE CIRCUIT

This circuit must provide a fixed output voltage of 2 V. The input voltage can vary inside the interval 4 V$\leq V_i \leq$6 V and

the temperature can vary in the interval 0 °C $\leq T \leq$ 100 °C. Candidate circuits will be simulated with NGSpice using a voltage DC sweep analysis in the first interval with a step size of 0.1 V and a temperature sweep analysis in the second interval with a step size of 25 °C, giving a total of 105 (21×5) fitting points. The threshold used here to consider a success is $V_{th} = 0.02$ V.

3) GAUSSIAN FUNCTION GENERATOR CIRCUIT

This circuit must generate an output current which is a Gaussian function of the input voltage. The fitness function is the same as that defined by equations (5) and (6), but using I_{o_i}, $\widetilde{I}_{o_i}$ and I_{th} instead of V_{o_i}, $\widetilde{V}_{o_i}$ and V_{th}, respectively. The input voltage can vary in the interval 2 V $\leq V_i \leq$ 3 V, and the peak of the Gaussian occurs at 2.5 V input with a standard deviation of 0.1 V. The output current peak is 80 nA. Candidate circuits will be simulated with NGSpice using a voltage DC sweep analysis in the mentioned interval with a step size of 0.01 V, giving a total of 101 fitting points. The threshold used here to consider a success is $I_{th} = 5$ nA.

4) COMPUTATIONAL CIRCUITS

The set of computational circuits comprise four circuits that compute a mathematical function of the input voltage value: square, square root, cubing and cube root. The input voltage varies in the interval −250 mV to 250 mV, except for the square root circuit that varies between 0 V and 500 mV. The output voltage must be consistent with the mathematical function implemented by each computational circuit.

The value used in [62] for V_{th} was very demanding and very few computational circuits met this requirement. In order to increment the number of successful circuits and facilitate the statistical comparison between methods, a less strict threshold was used here: instead of using $V_{th} = 1\%\ V_{o_i}$, we used $V_{th} = 5\%\ max(V_{o_i})$. Therefore, to do a fair comparison between the results obtained by the method presented here and that used in [62], the new threshold was used with both methods when the computational circuit design was addressed. Candidate circuits will be simulated with NGSpice using a time-domain analysis and a voltage ramp with a rise time of 0.2 s as the input signal. In this case, we do not select a DC sweep analysis because we assume that, in this type of circuit, the input could vary faster than that used for the non-computational circuits. Additionally, there is evidence in the literature that the evaluation of the fitness function based on a DC sweep analysis can lead to less robust circuit designs [19], [63].

The seven circuits proposed can be considered as challenging problems that have been tackled by different evolutionary paradigms in the related literature [13], [17], [62]–[65].

B. CONFIGURATION PARAMETERS

Table 3 shows the parameter configuration used in the MGE-based implemented method to address the seven analog circuits described in the previous section. Since we are interested in comparing the results of our method with those obtained in [62], where a GE-based method was used, we imposed the following restrictions to do a fair comparison: the GA-based search engine uses the same parameter configuration in both cases, except that a one-block crossover operator is used in [62] and the BG-MHX operator is used in our approach. Additionally, regarding the parameter that controls the maximum chromosome length, its value was selected to store the same maximum number of components (assuming that wrapping is not used) in both methods. In any case, there would still be room for improvement if the parameter tuning of our approach were customized for each analyzed circuit.

The MNN parameter requires special attention since its value can determine the success or not of a run. On the one hand, a low value reduces the search space, but it can prevent obtaining a solution; on the other hand, a high value can guarantee to obtain a solution, but not the optimal one (circuit with more nodes and possibly more components than necessary). For each circuit, we used the same MNN value as that used in [62] for the sake of a fair comparison. However, we propose two different strategies to adjust this parameter. The first one, called *direct strategy*, starts by selecting a low value for MNN and carrying out several executions. Then, if the number of hits reached by the best obtained circuit does not reach the maximum, the MNN value will be increased and new executions will be carried out. This cycle will be repeated until obtaining a circuit that achieves 100% of hits. The second strategy, called *indirect strategy*, acts on the fitness function definition, establishing a compromise between complying with the design specifications and minimizing the circuit complexity. In this case, the choice of the MNN value is less critical: the user can choose a large enough value and the evolutionary algorithm will look for the circuit that meets the specifications and has the minimum number of components (and, indirectly, the minimum number of nodes). We will return to this strategy in Section V-C.

V. RESULTS AND DISCUSSION

In this section, the results of applying our MGE-based method in each proposed benchmark circuit are shown. These results are compared with those obtained with a GE-based method. To simplify the notation, from now on, the implemented method based on MGE will be denoted by ACID-MGE (analog circuit design based on multi-grammatical evolution), and the GE-based method will be denoted by ACID-GE (analog circuit design based on grammatical evolution) [62]. Besides, we will compare the behavior of ACID-MGE with other evolutionary algorithms that have also addressed the design of the circuits proposed here. Finally, we analyze the ACID-MGE's behavior when a compromise between complexity and performance in the evolutionary search for the best solution circuit is contemplated.

Common performance measures of evolutionary algorithms such as the success rate (SR), mean best fitness (MBF) and minimum best fitness (minBF) are used. In particular, given an experiment comprising several runs, the SR is defined as the ratio of the number of successful runs

TABLE 3. Parameter configuration of the MGE-based method for the analog circuit design problem.

Circuit type	Temperature sensor, Voltage reference, Gaussian function or Computing circuits
Grammars	Set of two grammars shown in tables 1 and 2 adapted to the benchmark circuit
Search engine	GA with variable length chromosome
Fitness function	Depends on benchmark circuit (see Section IV-A)
Success criterion	When $\forall i \, \lvert V_{o_i} - \widetilde{V}_{o_i} \rvert \leq V_{th}$ (see Eqs. 5 and 6)
Population size	1000
Representation	Codon strings of variable length
Maximum number of generations (NG_{max})	3000
Initialization	Random codon strings of $150 - 250$ length
Maximum chromosome length	336 codons (42 components without using wrapping)
Crossover	BG-MHX operator (see Section III-D)
Crossover probability	0.5
Block size (component)	8 codons
Mutation	Bitwise
Mutation probability	0.001
Parent selection	Tournament selection (size=3)
Survival selection	Generational replacement
Elitism	2
Wrapping	4
Termination condition	Maximum number of generations
Random number generator	Mersenne Twister
Number of runs per circuit	50

concerning the total number of runs, where one success is obtained when an individual scores all possible hits. The MBF is the fitness average of the best individuals, each of which is obtained at the end of a run, regardless of whether a success was achieved or not. By last, the minBF is the best solution obtained in the set of runs associated with an experiment. To assess the statistical significance of the SR values obtained in each comparison, an one-tailed two-proportions z-test (with level significance $\alpha = 0.05$) is done, and the p-value is calculated. Here, we use the property that the sampling distribution of a difference between two proportions corresponds to a normal distribution when $n > 30$, being n the size of each sample. The number of components of the best circuit (NCBC) obtained is also shown to evaluate its complexity.

A. ACID-MGE VS. ACID-GE

Tables 4 and 5 show ACID-MGE *vs.* ACID-GE results for non-computational and computational circuits, respectively. Note that the values for the MNN parameter were the same as those used in [62]. In addition to the performance measures mentioned above, the mean average error (MAE) for the computational circuits is also shown. In particular, the MAE is calculated using (7), where O_i and $\widetilde{O}_i$ are the expected and measured output, respectively, obtained for the i-th fitting point.

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |O_i - \widetilde{O}_i| \tag{7}$$

Considering the SR, there is a statistically significant improvement (p-value < 0.05) in 5 out of 7 circuits. Additionally, though in 2 out of 7 circuits the null hypothesis cannot be rejected, the SR obtained by ACID-MGE was still equal to or greater than that obtained by ACID-GE. Therefore, we can say that ACID-MGE outperforms or equals to ACID-GE when the SR is considered. Regarding the MBF and mean hits, ACID-MGE outperformed ACID-GE in all the circuits tested. Considering the minBF (the best circuit obtained), there was also improvement in 6 out of 7 circuits tested. Both ACID-GE and ACID-MGE on the Gaussian function generator circuit obtained the same minBF. Lastly, ACID-MGE outperformed ACID-GE in all the computational circuits concerning the MAE. However, considering the complexity of the best circuits, ACID-GE outperforms ACID-MGE, that is, and more specifically, the NCBC obtained by our method is larger in 6 out of 7 circuits.

The measured outputs of the best circuits, obtained with the ACID-MGE algorithm, are compared with the expected ones. Figures 8a, 8b and 8c show, respectively, the output voltage of the best temperature sensor circuit, the output voltage of the best voltage reference circuit, and the output current of the best Gaussian function generator circuit. Additionally, Figs. 9a, 9b, 9c and 9d show, respectively, the output voltages of the best squaring, cubing, square root and cube root circuits obtained. As it can be seen, a good fit is obtained in all cases.

The convergence speed can also be analyzed. In particular, the mean number of generations needed to obtain a successful solution,Mean(#*Gen*), is also shown in tables 4 and 5. The smaller this index is, the faster the algorithm will be in finding a solution. As can be seen in these tables, MGE is faster in convergence than GE in all cases. The time for a run in ACID-MGE depends on the evolved circuit, the number of generations, and the hardware used. For example, for 3000 generations, the average execution time is 60-70 min, using a computer cluster of eight PCs: two of them with processor Core i5-7500@3.40 Ghz, two with processor Core i5-6500@3.2 Ghz and four more with processor Core 2 Quad@2.66 GHz.

B. ACID-MGE VS. OTHER EVOLUTIONARY METHODS

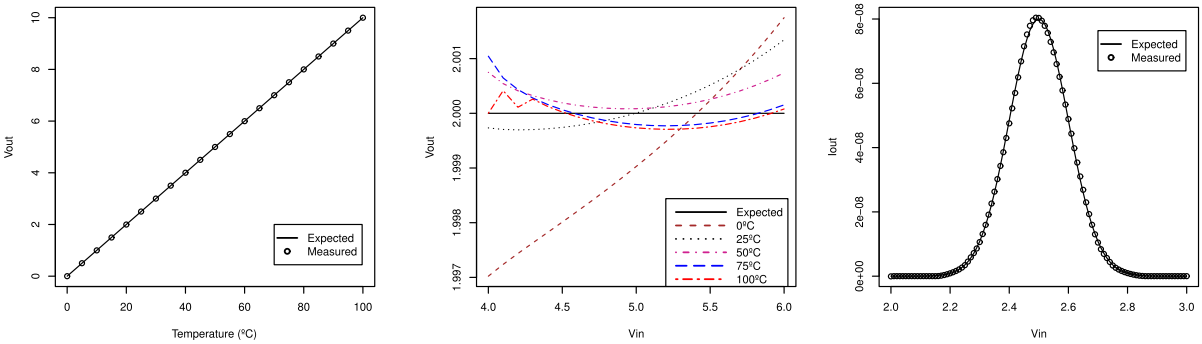
We compare our results with other evolutionary approaches that have also addressed the design of the proposed benchmark circuits. In particular, the three non-computational circuits were synthesized in previous works using GP [13] and *analog genetic encoding* (AGE) [17]. About the four computational circuits, they were synthesized using GP [13], [63], [64] and evolution strategies (ES) [19]. The cubing circuit is also compared with a conventional design [66]. Table 6 shows the results of the comparison for the best non-computational circuits. The results of the methods with which we compare are shown as presented in the papers where they were published and, to make a fair comparison, all the fitness values shown were computed with the same fitness function. Table 7 shows the results of the comparison for the best computational circuits. The results of the methods with

TABLE 4. ACID-MGE *vs.* ACID-GE for the non-computational circuits ($NG_{max} = 3000$). The standard deviation is denoted by SD. The best result for each circuit is marked in bold.

Circuit	Algorithm	MNN	SR(%)	p-value (SR)	minBF	MBF ±SD	Mean Hits(%) ±SD	Mean #Gen ±SD	NCBC
Temperature sensor	ACID-MGE	6	**70.0**	0.0016	**0.033**	**2.13 ±3.34**	**97.1 ±5.3**	**711.2 ±726.2**	42
	ACID-GE	6	42.0		0.169	5.29 ±6.53	93.0 ±7.9	1,278.9 ±884.6	**28**
Gaussian function	ACID-MGE	6	**58.0**	0.0001	**0.028**	**1.00 ±1.75**	**94.1 ±9.8**	**1,023.2 ±719.6**	**37**
	ACID-GE	6	24.0		0.040	6.70 ±6.78	77.3 ±17.6	1,367.4 ±688.5	41
Voltage reference	ACID-MGE	6	**8.0**	0.5000	**0.053**	**19.35 ±14.04**	**64.2 ±23.3**	**1,205.0 ±863.3**	42
	ACID-GE	6	**8.0**		0.112	26.57 ±16.23	53.8 ±25.0	1,581.0 ±917.0	**32**

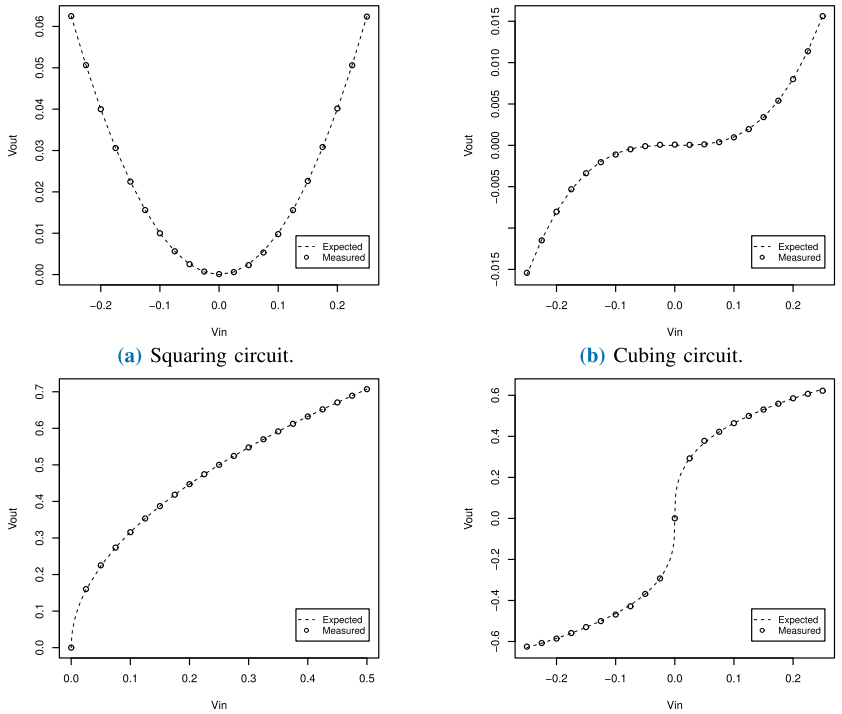
TABLE 5. ACID-MGE *vs.* ACID-GE for the computational circuits ($NG_{max} = 3000$). The standard deviation is denoted by SD. We have marked with bold letters the best results for each circuit.

Circuit	Algorithm	MNN	SR(%)	p-value (SR)	MAE ±SD	minBF	MBF ±SD	Mean Hits(%) ±SD	Mean #Gen ±SD	NCBC
Squaring	ACID-MGE	10	**84.0**	0.0001	**0.53 ±0.24**	**0.002**	**0.06 ±0.14**	**97.7 ±6.5**	**671.5 ±574.9**	42
	ACID-GE	10	52.0		0.93 ±0.41	0.009	0.73 ±1.17	81.0 ±28.5	1,176.9 ±828.3	**28**
Square root	ACID-MGE	10	**66.0**	0.0117	**4.01 ±2.47**	**0.003**	**2.71 ±5.78**	**89.0 ±24.2**	**1,154.9 ±655.0**	44
	ACID-GE	10	44.0		5.55 ±4.32	0.028	6.25 ±7.61	74.0 ±32.3	1,329.3 ±797.3	**35**
Cubing	ACID-MGE	20	**84.0**	0.1574	**0.10 ±0.04**	**0.001**	**0.03 ±0.08**	**95.7 ±13.7**	**539.8 ±361.7**	47
	ACID-GE	20	76.0		0.18 ±0.07	0.002	0.05 ±0.12	92.2 ±19.3	1,008.0 ±703.6	**40**
Cube root	ACID-MGE	30	**22.0**	0.0006	**7.13 ±3.70**	**0.036**	**13.87 ±25.02**	**70.2 ±29.4**	**1,561.1 ±587.8**	57
	ACID-GE	30	2.0		10.81 ±0.0	0.227	76.95 ±23.74	11.5 ±20.2	1,885.0 ±0.0	**41**



(a) Temperature sensor. (b) Voltage reference. Note the reduced scale of Y axis ($\triangle V_{out} = 2mV$). (c) Gaussian function generator.

FIGURE 8. Measured output versus expected output for the best non-computational circuits obtained with the ACID-MGE algorithm.



(a) Squaring circuit. (b) Cubing circuit.

(c) Square root circuit. (d) Cube root circuit.

FIGURE 9. Measured output versus expected output for the best computational circuits obtained with the ACID-MGE algorithm.

which we compare are shown as presented in [19]. In this case, the MAE is used, given that we cannot guarantee that the same fitness function was used in all the cases. As can be seen, the results obtained with MGE are competitive in the seven circuits analyzed regarding the fitness or MAE. However, as it already occurred in the comparison with ACID-GE, when complexity is considered, ACID-MGE is not the best compared to the referenced evolutionary methods. The only exception is the voltage reference circuit, with which our method achieves the least number of components.

TABLE 6. Comparison with previous methods for the best non-computational circuits. The best results are marked in bold.

	GP [13]	AGE [17]	ACID-MGE
Temperature sensor			
Fitness	26.49	1.13	**0.03**
NCBC	54	**28**	42
Voltage reference			
Fitness	6.60	2.64	**0.05**
NCBC	67	70	**42**
Gaussian function			
Fitness	0.09	0.30	**0.03**
NCBC	**14**	36	37

C. COMPLEXITY VS. PERFORMANCE IN ACID-MGE

Although the performances obtained by ACID-MGE in the seven analyzed circuits, regarding the minBF or MAE, surpass those obtained by ACID-GE and other evolutionary algorithms, the complexity of the best circuits obtained, regarding the NCBC, is generally worse (see Tables 6 and 7). This is an important point to consider because, in electronic circuit design, the proposed solutions must not only meet the design specifications but they should also find a compromise between complexity and performance. Therefore, with this in mind, we will now focus on analyzing whether ACID-MGE could obtain circuits that meet the specifications using as few components as possible.

TABLE 7. Comparison with previous methods for the best computational circuits. The best results are marked in bold.

	GP [13]	GP [63]	GP [64]	Manual design [66]	ES [19]	ACID-MGE
Square root						
MAE (mV)	183.57	20	-	-	9.23	**0.23**
NCBC	64	39	-	-	**22**	44
Squaring						
MAE (mV)	-	27	-	-	1.44	**0.08**
NCBC	39	37	-	-	**35**	42
Cube root						
MAE (mV)	80.00	-	-	-	11.90	**2.04**
NCBC	50	-	-	-	**39**	57
Cubing						
MAE (mV)	1.04	-	0.99	7.13	0.29	**0.05**
NCBC	56	-	47	**12**	44	47

When there are two or more objectives to optimize, a multi-objective approach is needed. In evolutionary computing, Pareto dominance based algorithms, like NSGA-II [67], are usually used in these cases. However, we propose here a simpler approach, called *simple parsimony*, that transforms the problem of optimizing two objectives into a new problem that consists of combining constraint satisfaction and objective optimization. In particular, the circuit specifications will be considered as restrictions to satisfy and, the number of components, as an objective to minimize. Additionally, we prioritize to meet the circuit specifications before to minimize the number of components, in such a way that the second objective will only be tackled when the first one is reached. To do this, we define the following fitness function:

$$newfitness = \begin{cases} f_1 & \text{if \#hits} < \text{MAX} \\ f_2 & \text{otherwise} \end{cases} \tag{8}$$

where f_1 is given by (5), MAX is a constant that defines the maximum number of hits that is possible to reach in the circuit to design, and f_2 is defined as:

$$f_2 = k \cdot NC \tag{9}$$

being k a scale factor, and NC the number of components of the evolved circuit. It is important to note that if #hits $=$ MAX, the values provided by *newfitness* must always be lower than those obtained when #hits $<$ MAX. In this way, the algorithm will always prioritize to meet the design specification before optimizing the number of components. Thus, with this strategy, given two circuits c_1 and c_2, if $newfitness(c_1) < newfitness(c_2)$, then we always guarantee one of the three following results: (i) c_1 and c_2 meet the design specifications, but c_1 has fewer components than c_2; (ii) c_1 and c_2 do not meet the design specifications, but c_1 reaches more hits than c_2; (iii) c_1 meets the design specifications, but c_2 does not. To implement this strategy, it is necessary to properly select the value of k. In particular, we consider the case in which a circuit has the maximum number of components ($maxNC$) that can be encoded in a chromosome (considering the wrapping mechanism). Then, in this case, f_2 must provide the value of f_1 when $\#hits = MAX - 1$ and all the absolute differences shown in (5) are zero except one of them, which takes the value $V_{th} + \epsilon$, with $\epsilon \to 0$. Then, in this situation, f_1 takes the value $f_1^* = \max(w_i) \times (V_{th} + \epsilon)$ according to (5) and (6), and k represents the slope of a line that passes through the origin and the point $(maxNC, f_1^*)$, that is:

$$k = \frac{f_1^*}{maxNC} \tag{10}$$

Tables 8 and 9 show the results obtained by ACID-MGE, depending on whether parsimony is used or not, for the computational and non-computational circuits, respectively. Regarding the NCBC obtained, it can be seen that ACID-MGE with parsimony can outstandingly reduce the circuit complexity compared to ACID-MGE without parsimony for the seven circuits analyzed. In the case of the cube root

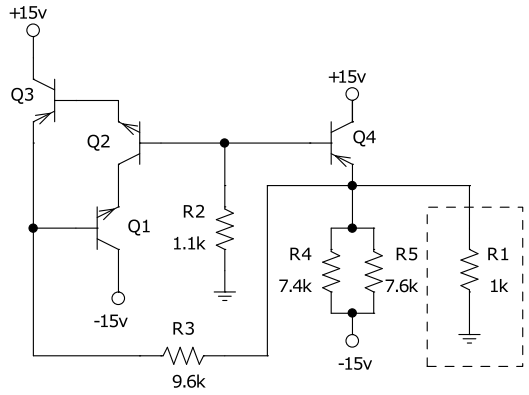


FIGURE 10. Best temperature sensor circuit obtained using ACID-MGE with parsimony (*MNN* = 10 and 3 000 generations).

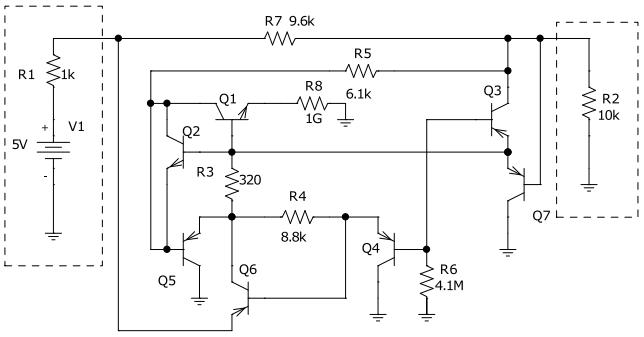


FIGURE 11. Best voltage reference circuit obtained using ACID-MGE with parsimony (*MNN* = 10 and 3 000 generations).

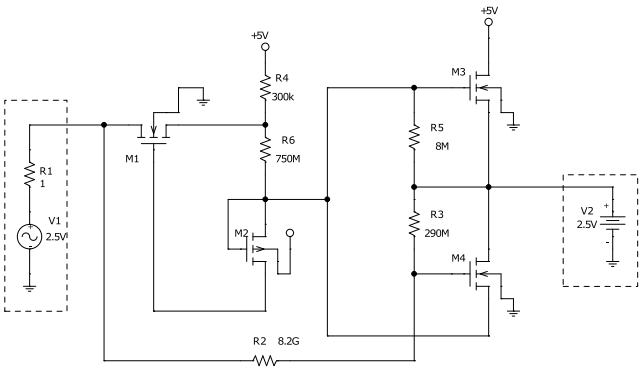


FIGURE 12. Best Gaussian function circuit obtained using ACID-MGE with parsimony (*MNN* = 10 and 3 000 generations).

circuit, more generations were needed (6, 000), and, additionally, it was needed to set $MNN = 15$. However, it must be taken into account that this circuit required the highest number of components. Besides, the complexity of the seven circuits obtained using ACID-MGE with parsimony is minor in comparison with ACID-GE (see Tables 4 and 5) and other evolutionary approaches (see Tables 6 and 7). Regarding the SR and considering the p-value, ACID-MGE with parsimony behaves the same (6/7 cases) as or better (1/7 cases) than ACID-MGE without parsimony. Note that this

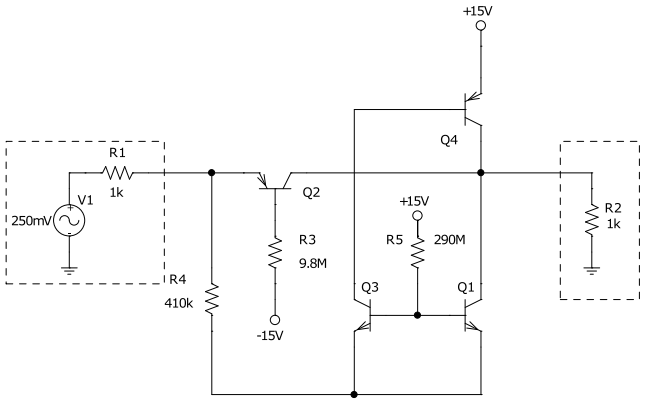


FIGURE 13. Best squaring circuit obtained using ACID-MGE with parsimony (*MNN* = 10 and 3 000 generations).

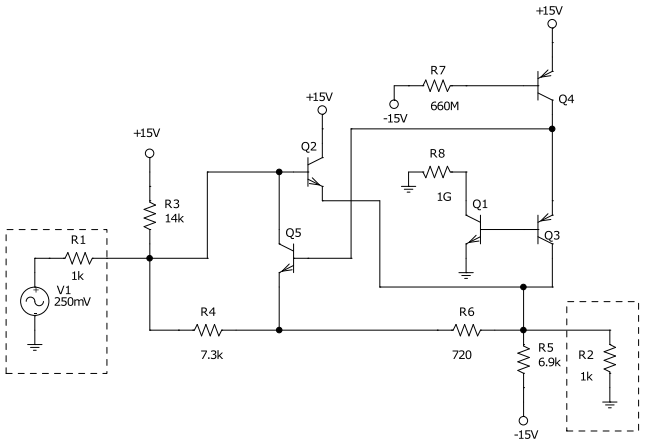


FIGURE 14. Best square root circuit obtained using ACID-MGE with parsimony (*MNN* = 10 and 3 000 generations).

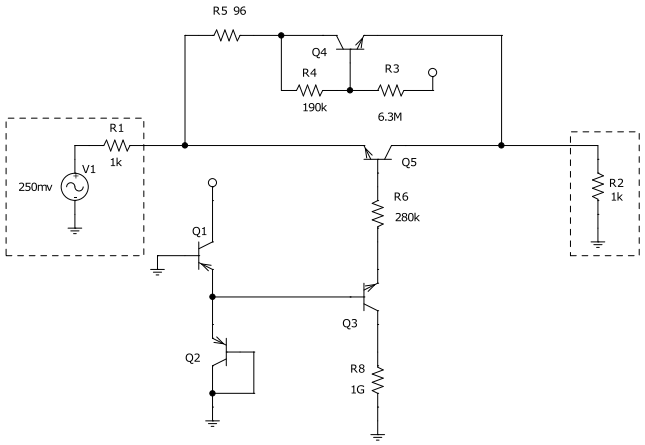


FIGURE 15. Best cubing circuit obtained using ACID-MGE with parsimony (*MNN* = 20 and 3 000 generations). In this schematic, a resistor that connected the negative supply −15*V* with ground was removed, since it did not make any function in the circuit.

result is somewhat surprising since the task of meeting the design specifications with the fewest number of components

TABLE 8. Comparison of the SR and the number of components of the best circuit (NCBC) obtained using ACID-MGE when simple parsimony (PAR) is used or not for the non-computational circuits.

Circuit	PAR	MNN	# Gen.	SR	p-value (SR)	NCBC
Temperature sensor	Yes	10	3,000	60.0%	0.0062	**8**
	No	10	3,000	**82.0%**		50
Gaussian function	Yes	10	3,000	66.0%	0.3339	**9**
	No	10	3,000	**70.0%**		44
Voltage reference	Yes	10	3,000	20.0%	0.4030	**13**
	No	10	3,000	**22.0%**		42

TABLE 9. Comparison of the SR and the number of components of the best circuit (NCBC) obtained using ACID-MGE when simple parsimony (PAR) is used or not for the computational circuits.

Circuit	PAR	MNN	# Gen.	SR	p-value (SR)	NCBC
Squaring	Yes	10	3,000	**88.0%**	0.2819	**7**
	No	10	3,000	84.0%		42
Square root	Yes	10	3,000	**76.0%**	0.1338	**11**
	No	10	3,000	66.0%		44
Cubing	Yes	20	3,000	**88.0%**	0.2819	**10**
	No	20	3,000	84.0%		47
Cube root	Yes	15	6,000	**32.0%**	-	**22**
	Yes	30	3,000	18.0%	0.3083	50
	No	30	3,000	22.0%		57

TABLE 10. Channel width and MOSFET type for the best Gaussian function circuit (see schematic in figure 12) obtained by ACID-MGE using parsimony.

Transistor	Type	Channel width (μm)
M1	n-channel	121
M2	p-channel	31
M3	n-channel	108
M4	n-channel	56

is harder than the task of enforcing only the design specifications.

Finally, Figs. 10, 11, and 12 show the schematics obtained for ACID-MGE with parsimony for the best temperature sensor, voltage reference, and Gaussian function circuits, respectively. Regarding the computational circuits, Figs. 13, 14, 15, and 16 show the schematics for the best squaring, square root, cubing, and cube root circuits, respectively. Additionally, table 10 shows the MOSFET type and the channel width for the MOSFETs used in the best Gaussian function circuit shown in Fig. 12.

VI. CONCLUSION

In this paper, we have presented MGE, a new evolutionary approach used to address the analog electronic circuit design problem. MGE is a variant of GE that uses modularity

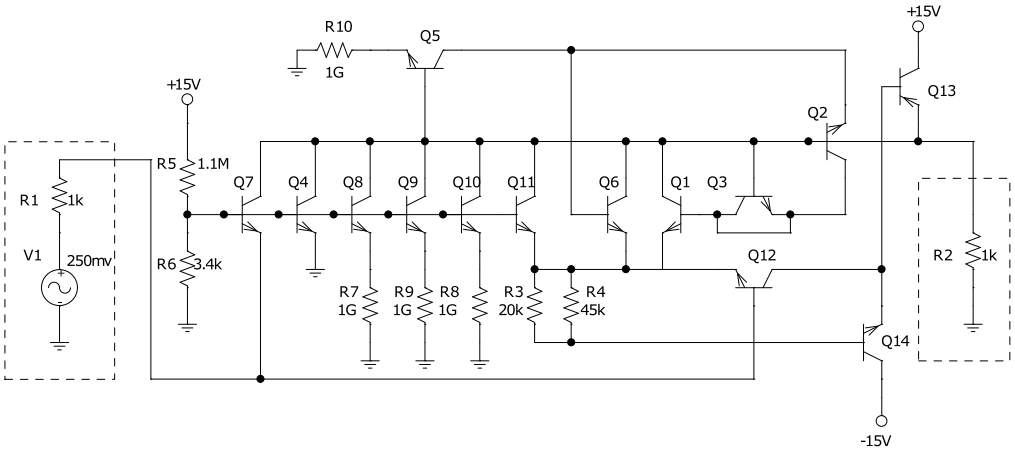


FIGURE 16. Best cube root circuit obtained using ACID-MGE with parsimony (*MNN* = 15 and 6 000 generations).

and homology. Modularity allows us to divide the original problem into different subproblems in such a way that the solution to each subproblem is encoded and evolved in different partitions of the chromosome, and each of them is decoded according to a different grammar. The final solution associated with each chromosome is obtained as a composition of the partial solutions of each partition. In another way, homology helps to build crossover operators that reduce the potentially destructive effect associated with standard crossover operators traditionally used in GE.

An MGE-based method was implemented and applied to the design of seven analog electronic circuits. Each problem is decomposed into two levels of abstraction (topology and sizing) and each level is formalized by a specific grammar. In this context, regarding the SR and minBF, we provide evidence that the MGE-based method is more effective and efficient than a GE-based method, that is, the first method is capable of improving the results obtained by the second and with a lower number of generations. In order to do a fair comparison between both approaches, we used, for each problem, the same configuration of parameters in the GA used as a search engine. We compare our approach with other evolutionary approaches (GP, ES, and AGE) and the obtained results were also competitive in the seven circuits analyzed. However, the MGE-based method tends to provide circuits more complex (larger number of components). A simple parsimony approach was implemented to simultaneously address the task of meeting the design specifications and minimize the number of components of the evolved circuit. In this context, interesting results were obtained: the SR was maintained, but the number of components were significantly reduced in the best circuits obtained for the seven proposed benchmark circuits.

Although MGE has shown promising results in the analog circuit design problem, more research is needed to assess its behavior, for example, in more demanding analog circuit design or in other application domains. In order to preserve the homology property, MGE is more appropriate for addressing hierarchically decomposable design problems, given that, in this kind of problem, it is easier to define grammars that support recursiveness on the right, specially in the rule that defines the components to be utilized in each level of abstraction. Future research could also focus on the automatic design of artificial neural networks, where it is also easy to divide the original problem into different levels: network topology (number and types of layers and number and types of neurons by layer) and connection weights.

DECLARATION OF INTERESTS

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

REFERENCES

[1] D&R. (2018). *Analog IC Market Forecast With Strongest Annual Growth Through 2022*. Accessed: Apr. 26, 2020. [Online]. Available: https://www.design-reuse.com/news/43374/analog-ic-market-forecast.html

[2] H. Camenzind, *Designing Analog Chips*. College Station, TX, USA: Virtual Bookworm, 2005.

[3] G. G. E. Gielen and R. A. Rutenbar, "Computer-aided design of analog and mixed-signal integrated circuits," in *Computer-Aided Design of Analog Integrated Circuits and Systems*, R. A. Rutenbar, G. G. E. Gielen, and B. A. Antao, Eds. New York, NY, USA: Wiley, 2002, pp. 3–10.

[4] O. Aaserud and I. R. Nielsen, "Trends in current analog design—A panel debate," *Anal. Integr. Circuits Signal Process.*, vol. 7, no. 1, pp. 5–9, Jan. 1995.

[5] S. E. Sorkhabi and L. Zhang, "Automated topology synthesis of analog and RF integrated circuits: A survey," *Integration*, vol. 56, pp. 128–138, Jan. 2017.

[6] J. Scheible and J. Lienig, "Automation of analog IC layout: Challenges and solutions," in *Proc. Symp. Int. Symp. Phys. Design (ISPD)*, New York, NY, USA, 2015, pp. 33–40.

[7] R. A. Rutenbar, G. G. E. Gielen, and J. Roychowdhury, "Hierarchical modeling, optimization, and synthesis for system-level analog and RF designs," *Proc. IEEE*, vol. 95, no. 3, pp. 640–669, Mar. 2007.

[8] B. Gilbert, "Analog design in the information age," in *Proc. BIPOLAR/BiCMOS Circuits Technol. Meeting*, Oct. 2001, pp. 118–123.

[9] R. S. Zebulum, M. A. Pacheco, and M. Vellasco, "Comparison of different evolutionary methodologies applied to electronic filter design," in *Proc. IEEE Int. Conf. Evol. Comput., IEEE World Congr. Comput. Intell.*, May 1998, pp. 434–439.

[10] J. R. Koza, F. H. Bennett, J. Lohn, F. Dunlap, M. A. Keane, and D. Andre, "Automated synthesis of computational circuits using genetic programming," in *Proc. IEEE Int. Conf. Evol. Comput. (ICEC)*, Apr. 1997, pp. 447–452.

[11] J. R. Koza, F. H. Bennett, D. Andre, and M. A. Keane, "Evolution using genetic programming of a low-distortion, 96 decibel operational amplifier," in *Proc. ACM Symp. Appl. Comput. (SAC)*, New York, NY, USA, 1997, pp. 207–216.

[12] J. Koza, F. Bennett, III, D. Andre, and M. Keane, "The design of analogue circuits by means of genetic programming," in *Evolutionary Design by Computers*, P. J. Bentley, Ed. Hoboken, NJ, USA: Wiley, 1999, ch. 16, pp. 365–385.

[13] J. Koza, D. Andre, F. Bennett, III, and M. Keane, *Genetic Programming III: Darwinian Invention and Problem Solving*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann, 1999.

[14] J. B. Grimbleby, "Automatic analogue network synthesis using genetic algorithms," in *Proc. 1st Int. Conf. Genet. Algorithms Eng. Syst., Innov. Appl. (GALESIA)*, 1995, pp. 53–58.

[15] A. Thompson, "Artificial evolution in the physical world," in *Evolutionary Robotics: From Intelligent Robotics to Artificial Life (ER)*, T. Gomi, Ed. Kanata, ON, Canada: AAI Books, 1997, pp. 101–125.

[16] S. Ando and H. Iba, "Analog circuit design with a variable length chromosome," in *Proc. Congr. Evol. Comput. (CEC)*, vol. 2, 2000, pp. 994–1001.

[17] C. Mattiussi and D. Floreano, "Analog genetic encoding for the evolution of circuits and networks," *IEEE Trans. Evol. Comput.*, vol. 11, no. 5, pp. 596–607, Oct. 2007.

[18] E. Tlelo-Cuautle and M. Duarte-Villaseñor, "Evolutionary electronics: Automatic synthesis of analog circuits by GAs," in *Success in Evolutionary Computation*, vol. 92, A. Yang, Y. Shan, and L. Bui, Eds. Berlin, Germany: Springer, 2008, pp. 165–187.

[19] Y. A. Sapargaliyev and T. G. Kalganova, "Open-ended evolution to discover analogue circuits for beyond conventional applications," *Genet. Program. Evolvable Mach.*, vol. 13, no. 4, pp. 411–443, Dec. 2012.

[20] J. Slezák and J. Petržela, "Evolutionary synthesis of cube root computational circuit using graph hybrid estimation of distribution algorithm," *Radioengineering*, vol. 23, no. 1, p. 549, 2014.

[21] V. Z. Rojec, A. Bűrmen, and I. Fajfar, "Analog circuit topology synthesis by means of evolutionary computation," *Eng. Appl. Artif. Intell.*, vol. 80, pp. 48–65, Apr. 2019.

[22] M. Kunaver, "Grammatical evolution-based analog circuit synthesis," *J. Microelectron., Electron. Compon. Mater.*, vol. 49, no. 4, pp. 229–240, 2020.

[23] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing* (Natural Computing Series). Berlin, Germany: Springer, 2003.

[24] T. McConaghy, P. Palmers, M. Steyaert, and G. G. E. Gielen, "Variation-aware structural synthesis of analog circuits via hierarchical building blocks and structural homotopy," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 9, pp. 1281–1294, Sep. 2009.

[25] M. J. Streeter, J. R. Koza, and M. A. Keane, "Routine automated synthesis of five patented analog circuits using genetic programming," *Soft Comput.*, vol. 8, no. 5, pp. 318–324, Apr. 2004, doi: 10.1007/s00500-003-0288-9.

[26] A. Stoica, R. S. Zebulum, X. Guo, D. Keymeulen, M. I. Ferguson, and V. Duong, "Taking evolutionary circuit design from experimentation to implementation: Some useful techniques and a silicon demonstration," *IEE Proc.-Comput. Digit. Techn.*, vol. 151, no. 4, pp. 295–300, Jul. 2004.

[27] T. McConaghy and G. Gielen, "Canonical form functions as a simple means for genetic programming to evolve human-interpretable functions," in *Proc. 8th Annu. Conf. Genet. Evol. Comput. (GECCO)*, 2006, pp. 855–862.

[28] P. Nenzi and H. Vogt. (2011). *NGSPICE User's Manual Version 23*. [Online]. Available: http://ngspice.sourceforge.net/

[29] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, vol. 1. Cambridge, MA, USA: MIT Press, 1992.

[30] A. Cano, S. Ventura, and K. J. Cios, "Multi-objective genetic programming for feature extraction and data visualization," *Soft Comput.*, vol. 21, no. 8, pp. 2069–2089, Apr. 2017.

[31] V. K. Dabhi and S. Chaudhary, "Empirical modeling using genetic programming: A survey of issues and approaches," *Natural Comput.*, vol. 14, no. 2, pp. 303–330, Jun. 2015.

[32] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proc. Genet. Evol. Comput. Conf.*, Jul. 2017, pp. 497–504.

[33] R. Aguilar-Rivera, M. Valenzuela-Rendón, and J. J. Rodríguez-Ortiz, "Genetic algorithms and darwinian approaches in financial applications: A survey," *Expert Syst. Appl.*, vol. 42, no. 21, pp. 7684–7697, Nov. 2015.

[34] I. González-Taboada, B. González-Fonteboa, F. Martínez-Abella, and J. L. Pérez-Ordóñez, "Prediction of the mechanical properties of structural recycled concrete using multivariable regression and genetic programming," *Construct. Building Mater.*, vol. 106, pp. 480–499, Mar. 2016.

[35] N. Lourenço, J. Ferrer, F. B. Pereira, and E. Costa, "A comparative study of different grammar-based genetic programming approaches," in *Proc. Eur. Conf. Genet. Program.* Cham, Switzerland: Springer, 2017, pp. 311–325.

[36] R. I. McKay, N. X. Hoai, P. A. Whigham, Y. Shan, and M. O'Neill, "Grammar-based genetic programming: A survey," *Genet. Program. Evolvable Mach.*, vol. 11, nos. 3–4, pp. 365–396, Sep. 2010.

[37] C. Ryan, J. J. Collins, and M. O'Neill, "Grammatical evolution: Evolving programs for an arbitrary language," in *Proc. 1st Eur. Workshop Genet. Program.*, in Lecture Notes in Computer Science, vol. 1391, W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty, Eds. Berlin, Germany: Springer-Verlag, 1998, pp. 83–96.

[38] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Trans. Evol. Comput.*, vol. 5, no. 4, pp. 349–358, Aug. 2001.

[39] W. Banzhaf, G. Beslon, S. Christensen, J. A. Foster, F. Képès, V. Lefort, J. F. Miller, M. Radman, and J. J. Ramsden, "From artificial evolution to computational evolution: A research agenda," *Nature Rev. Genet.*, vol. 7, no. 9, pp. 729–735, Sep. 2006.

[40] A. Merlevede, H. Åhl, and C. Troein, "Homology and linkage in crossover for linear genomes of variable length," *PLoS ONE*, vol. 14, no. 1, 2019, Art. no. e0209712.

[41] H.-J. Li, Q. Wang, S. Liu, and J. Hu, "Exploring the trust management mechanism in self-organizing complex network based on game theory," *Phys. A, Stat. Mech. Appl.*, vol. 542, Mar. 2020, Art. no. 123514.

[42] D. Fagan, M. O'Neill, E. Galván-López, A. Brabazon, and S. McGarraghy, "An analysis of genotype-phenotype maps in grammatical evolution," in *Proc. Eur. Conf. Genet. Program.* Berlin, Germany: Springer, 2010, pp. 62–73.

[43] A. Brabazon and M. O'Neill, "Credit rating with pi grammatical evolution," in *Proc. Comput. Methods Syst. Conf.*, vol. 1, R. Tadeusiewicz, A. Ligeza, and M. Szymkat, Eds. Kraków, Poland: Oprogramowanie Naukowo-Techniczne Tadeusiewicz, Nov. 2005, pp. 253–260.

[44] D. R. White, J. McDermott, M. Castelli, L. Manzoni, B. W. Goldman, G. Kronberger, W. Jaśkowski, U.-M. O'Reilly, and S. Luke, "Better GP benchmarks: Community survey results and proposals," *Genet. Program. Evolvable Mach.*, vol. 14, no. 1, pp. 3–29, Mar. 2013.

[45] P. A. Whigham, G. Dick, J. Maclaurin, and C. A. Owen, "Examining the 'best of both worlds' of grammatical evolution," in *Proc. Annu. Conf. Genet. Evol. Comput.*, 2015, pp. 1111–1118.

[46] C. Ryan, "A rebuttal to Whigham, Dick, and Maclaurin by one of the inventors of grammatical evolution: Commentary on 'on the mapping of genotype to phenotype in evolutionary algorithms' by Peter A. Whigham, Grant Dick, and James Maclaurin," *Genet. Program. Evolvable Mach.*, vol. 18, no. 3, pp. 385–389, 2017.

[47] F. D. Francone, M. Conrads, W. Banzhaf, and P. Nordin, "Homologous crossover in genetic programming," in *Proc. 1st Annu. Conf. Genet. Evol. Comput.*, vol. 2. San Francisco, CA, USA: Morgan Kaufmann, 1999, pp. 1021–1026.

[48] W. B. Langdon, "Size fair and homologous tree crossovers for tree genetic programming," *Genet. Program. Evolvable Mach.*, vol. 1, nos. 1–2, pp. 95–119, 2000.

[49] M. O'Neill, C. Ryan, M. Keijzer, and M. Cattolico, "Crossover in grammatical evolution," *Genet. Program. Evolvable Mach.*, vol. 4, pp. 67–93, Mar. 2003.

[50] R. S. Pressman, *Software Engineering: A Practitioner's Approach*. New York, NY, USA: McGraw-Hill, 1997.

[51] M. E. J. Newman, "Modularity and community structure in networks," *Proc. Nat. Acad. Sci. USA*, vol. 103, no. 23, pp. 8577–8582, Jun. 2006.

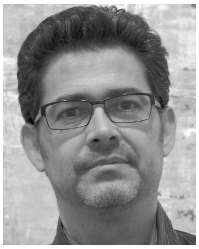
[52] H.-J. Li and J. J. Daniels, "Social significance of community structure: Statistical view," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 91, no. 1, Jan. 2015, Art. no. 012801.

[53] H.-J. Li, Z. Bu, Z. Wang, J. Cao, and Y. Shi, "Enhance the performance of network computation by a tunable weighting strategy," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 2, no. 3, pp. 214–223, Jun. 2018.

[54] G. P. Wagner and L. Altenberg, "Perspective: Complex adaptations and the evolution of evolvability," *Evolution*, vol. 50, no. 3, pp. 967–976, Jun. 1996.

[55] C. Ferreira, "Gene expression programming: A new adaptive algorithm for solving problems," *Complex Syst.*, vol. 13, no. 2, pp. 1–22, 2001.
[56] E. Clemente, G. Olague, D. Hernández, J. L. Briseño, and J. Mercado, "Object detection in natural images using the brain programming paradigm with a multi-objective approach," in *Applications of Evolutionary Computation*, A. M. Mora and G. Squillero, Eds. Cham, Switzerland: Springer, 2015, pp. 201–213.
[57] M. O'Neill and A. Brabazon, "Grammatical differential evolution," in *Proc. Int. Conf. Artif. Intell. (ICAI)*, H. R. Arabnia, Ed., vol. 1. Las Vegas, NV, USA: CSREA Press, 2006, pp. 231–236.
[58] M. O'Neill and A. Brabazon, "Grammatical swarm," in *Genetic and Evolutionary Computation*, K. Deb, Ed. Berlin, Germany: Springer, 2004, pp. 163–174.
[59] M. O'Neill and A. Brabazon, "Grammatical swarm: The generation of programs by social programming," *Natural Comput.*, vol. 5, no. 4, pp. 443–462, Nov. 2006, doi: 10.1007/s11047-006-9007-7.
[60] *Information Technology—Syntactic Metalanguage—Extended BNF*, Standard ISO/IEC-14977, 1996, doi: 10.3403/01300022.
[61] M. Nicolau and I. Dempsey, "Introducing grammar based extensions for grammatical evolution," in *Proc. IEEE Int. Conf. Evol. Comput.*, Jul. 2006, pp. 648–655.
[62] F. Castejón and E. J. Carmona, "Automatic design of analog electronic circuits using grammatical evolution," *Appl. Soft Comput.*, vol. 62, pp. 1003–1018, Jan. 2018.
[63] W. Mydlowec and J. Koza, "Use of time-domain simulations in automatic synthesis of computational circuits using genetic programming," in *Proc. Late Breaking Papers Genet. Evol. Comput. Conf.*, 2000, pp. 187–197.
[64] M. J. Streeter, M. A. Keane, and J. R. Koza, "Iterative refinement of computational circuits using genetic programming," in *Proc. Genet. Evol. Comput. Conf.* San Mateo, CA, USA: Morgan Kaufmann, 2002, pp. 877–884.
[65] Y. Sapargaliyev, "Automatic design of analogue circuits," Ph.D. dissertation, School Eng. Des., Brunel Univ., Uxbridge, U.K., 2011.
[66] S. Cipriani and A. Takeshian, "Compact cubic function generator," U.S. Patent 6 160 427, Dec. 12, 2000.
[67] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," in *Proc. Int. Conf. Parallel Problem Solving From Nature*. Berlin, Germany: Springer, 2000, pp. 849–858.

FEDERICO CASTEJÓN received the M.S. degree in telecommunications engineering from the Universidad Politécnica of Madrid, Spain, in 1992. He is currently pursuing the Ph.D. degree in intelligent systems with the Department of Artificial Intelligence, Universidad Nacional de Educación a Distancia (UNED), Madrid, Spain. He currently works at the General Secretariat of Digital Administration in the Ministry of Economic Affairs and Digital Transformation of Spain. His research interests include evolutionary computation, electronic circuit design, artificial intelligence, robotics, and data science.



ENRIQUE J. CARMONA received the M.S. degree in electronic engineering from the University of Granada, Spain, in 1996, and the Ph.D. degree in physics from the Universidad Nacional de Educación a Distancia (UNED), Madrid, Spain, in 2003. Since 2009, he has been an Associate Professor with the Department of Artificial Intelligence, School of Computer Engineering, UNED. He has authored or coauthored more than 20 peer-reviewed journal and conference papers. His research interests include computer vision, evolutionary computation, and the application of the latter to different areas (artificial vision, aeronautics, electronics, and mathematics).