Journal of Microwaves, Optoelectronics and Electromagnetic Applications, Vol. 22, No. 1, March 2023
DOI: http://dx.doi.org/10.1590/2179-10742023v22i1265775

Analog Electronic Circuit Synthesis Using Simulated Annealing and Geometric Circuit Evolution

Leonardo Muttoni, Antônio C. P. Veiga
Universidade Federal de Uberlândia (UFU), Av. João Naves de Ávila, 2121, Uberlândia, MG, Brazil
muttoni@gmail.com, acpveiga@gmail.com

***Abstract*— This article presents the SANN-GCE algorithm, a spice simulation driven meta-heuristic to design general discrete analog electronic circuits automatically, both circuit topology and component sizing. We introduce an encoding scheme called Geometric Circuit Evolution (GCE) that works associated with the Simulated Annealing algorithm and uses categorized degrees of freedom, that allows distinct characteristics of a circuit to change with different probabilities according to its type during the circuit evolution. We show through a series of seven active test circuits that SANN-GCE, compared to a benchmark, present a median fitness 15.88 times better, with a median standard deviation 6.72 times lower between runs. The median runtime found was 14.17 times lower.**

***Index Terms*—** Analog circuit synthesis, Automatic design, Computer Aided Design, Metaheuristic

I. Introduction

The manual design of analog electronic circuits requires expert knowledge and many iterations of testing and optimization until a circuit with the desired characteristics is obtained.

Electronic Design Automation (EDA) software tools are commonly used for schematic capture, circuit simulation and layout, helping the designer to reach a good solution in less time. Circuit schematics are typically created based on expert knowledge. Its increasingly stringent design requirements result in more complex circuits, a problem that could be alleviated with an additional level of automation: automatic circuit synthesis.

The problem of automatically constructing a circuit can be divided in two parts:

1) Topology synthesis: determines the quantity and types of electronic components in circuit, as well their interconnections. It is a very difficult combinatorial problem, as the search space is huge and a simple change of a single variable like the connection of a component terminal can completely alter the circuit behavior.
2) Sizing: involves determining the parameters of all circuit components, e.g, the value of resistance in resistors. They are usually continuous parameters of the components previously arranged in a circuit. Some commercial EDA tools have the ability to automatically optimize the sizing based on user specified desired performance target.

This subject is focus of several researches, specially the problem of topology synthesis. According to the extensive survey done in [1], early works on this subject mainly used human and knowledge-based topology selection [2], [3]. More recent works were categorized as being of type topology refinement

Brazilian Microwave and Optoelectronics Society-SBMO *received 7 July 2022; for review 11 July 2022; accepted 5 Jan 2023*
Brazilian Society of Electromagnetism-SBMag *ISSN 2179-1074*

or topology generation. In the first one, the method modifies a given initial circuit to improve its performance [4]. In the second, a topology is generated from scratch [5].

The contributions of [6] and [7] are considered a milestone for presenting a general technique to approach the problem of analog electronic circuit synthesis. It used Genetic Programming (GP) to evolve both topology and component values, creating an entire circuit from a high-level statement of the circuit's desired behavior.

In the following years, several techniques emerged in this matter, among which we cite the work of [8], that showed an encoding inspired in biological genetic regulatory networks that can be used to evolve analog circuits. It was tested in three analog circuit problems, using a genetic algorithm as the search method.

The reference [9] presented a system that uses GP to create analog circuits using a large database of trusted building blocks prepared previously by a human expert to tackle a specific problem.

Reference [10] used Grammatical Evolution (GE) [11] to express a candidate solution through a Backus-Naur Form grammar definition. The solutions are then evolved using a genetic algorithm. In another article, the authors presented the Multi Grammatical Evolution (MGE) [12]. It separated the original problem into two distinct sub-problems: component sizing and circuit topology. The algorithm was tested in seven use cases.

The authors in [13] proposed a technique that uses a connection matrix to represent the topology and optimize it together with the component values by means of an evolutionary algorithm, with crossover and mutation operations. It was demonstrated using three test circuits.

This paper presents Geometric Circuit Evolution (GCE), an encoding scheme that represents candidate solutions to the analog circuit synthesis problem. GCE introduces the concept of categorized Degree of Freedom (DOF) in the evolution of the circuit. In short, this allows different characteristics of a circuit to mutate with different probabilities according to its type. The GCE was implemented together with the classic search algorithm Simulated Annealing, that drives the evolution using the mutation of DOFs. The composition of GCE with Simulated Annealing is referred to hereinafter in this article as SANN-GCE.

The sole input of SANN-GCE is a formulation that specify the desired circuit behavior, and the output is a circuit designed at the discrete component level, described as a netlist. During algorithm execution, all candidate solutions are evaluated through Spice simulations.

As other algorithms that does not bound the search space, the circuits that can be automatically designed in SANN-GCE can assume virtually any topology, enabling the creation of unusual projects that may, by chance, be novel. This is in contrast to other techniques that uses expert information, like databases of pre-selected circuits block.

The main contributions of this article are: (1) we show how this novel algorithm (GCE) combined with classic simulated annealing can be used to successfully synthesize analog electronic circuits using a simpler mutation-only approach, evolving one solution at time (in contrast with population based algorithms, e.g. Genetic Algorithm); (2) its performance, when compared with a benchmark [12], has significant advantages in some key metrics: (2.1) it is $14.17\times$ faster; (2.2) the fitness of the best synthesized circuits is $15.88\times$ lower (better); (2.3) the fitness standard deviation of the best obtained circuits is $6.72\times$ lower.

The following sections are organized as follows: First, Section II explains the software framework

Brazilian Microwave and Optoelectronics Society-SBMO *received 7 July 2022; for review 11 July 2022; accepted 5 Jan 2023*
Brazilian Society of Electromagnetism-SBMag *ISSN 2179-1074*

and its modular parts. Then the Section III presents the simulated annealing algorithm, followed by Section IV that describes the Geometric Circuit Evolution, a proposed new solution representation that encodes a candidate solution in a convenient format for the simulated annealing to work efficiently. Next, the Sections V and VI shows how a problem should be designed in SANN-GCE and how its spice simulation works. It is presented in Sections VII and VIII seven test circuits as use cases for the algorithm, followed by a discussion of its results and some comparisons with other algorithms. In Section IX the final remarks of this work was reported.

II. Analog Electronic Circuit Synthesis Framework

A framework called `circ_autoproj` was built from scratch to run the optimizations. The software was written in C++ using its object-oriented paradigm. The framework has a modular architecture, with two main parts loaded at runtime as plugins according to a user configuration: 1) Search Algorithm (SA); 2) Solution Representation (SR). These parts have a standardized interface that allows for a seamless exchange of plugins, thus providing agility for comparing different algorithms.

The role of SR is to construct candidate solutions objects. Each object has a method to get its fitness, and methods to modify it, like the mutation operator. The SR itself loads another plugin at runtime: the problem. It contains specifications of the desired characteristics of the electronic circuit to be automatically designed, such as the spice analysis type, the desired input/output relation, its fitness function and the test fixture.

SA is normally a meta-heuristic algorithm that holds SR candidate solutions and evolve them over time, invoking methods provided by the SR. This is the part that drives the optimization.

The framework asks SA periodically to get the best solution found so far, and record this along with various statistics in an output file. This modular construction allow decoupling the parts, and helps the developer to improve one independently of the other.

In this article, the `circ_autoproj` has been configured to use a simulated annealing algorithm as the SA, and the GCE as the SR.

III. Search Algorithm: the Simulated Annealing

Simulated Annealing [14] is a well known classic meta-heuristic for global optimization that operates on one candidate solution at a time (in contrast with population based algorithms, like Genetic Algorithm). It is suitable for applications where objective function evaluation is complex and not explicitly known (black box problems). Its main advantages are simplicity and low memory usage, as it operates on one candidate solution at a time [15].

Its operation is based on an analogy with the annealing process in solids, which comprises heating the material and then slow cooling it in a controlled way. This process allows the material domains to be reorganized into a low energy state. The statistical mechanics unveil the theory behind this physical process, and it states that the probability of the solid being in an energy state $E_s = E$ is given by (1), where $Z(T)$ is a normalization factor, T is the temperature and k_B is the Boltzmann constant [16].

$$P(E_s = E) = \frac{1}{Z(T)} e^{-\frac{E}{k_B T}} \tag{1}$$

The Simulated Annealing uses the Metropolis criterion [17] to evolve its solution. This states the probability of accepting a new solution according to its fitness difference from the current solution

Brazilian Microwave and Optoelectronics Society-SBMO *received 7 July 2022; for review 11 July 2022; accepted 5 Jan 2023*
Brazilian Society of Electromagnetism-SBMag *ISSN 2179-1074*

($\Delta F = F_{new} - F_{curr}$). If the new solution is better, keep it. If it is worse, it will be accepted with a probability of $e^{-\Delta F/T}$, otherwise the current solution is maintained. This probability of acceptance is based on Boltzmann distribution, related to aforementioned physical annealing process [18].

The Simulated Annealing algorithm used in this work is shown in the Algorithm 1. It starts with a higher temperature $T = T_0$ and a random candidate solution S_0. At each iteration a neighbor solution S_1 is created from a mutation from S_0. If S_1 is better than S_0, makes the S_1 the current solution, decrease the temperature T by a factor α and proceed to the next iteration. If it is worse, uses the aforementioned Metropolis criterion to decide its acceptance.

Note that the higher the temperature (present in the start of the run), more likely a worse solution will be accepted. As the iterations advances, this probability fall as the temperature decreases. When T approaches zero, the Simulated Annealing becomes similar to the Monte Carlo algorithm [15].

The cooling schedule used in Algorithm 1 was the geometric cooling rule, given by $T_{new} = \alpha T_{old}$, where $0 < \alpha < 1$ is a constant called cooling factor usually very close to 1. Additionally, when the temperature drops below the reheating threshold T_r, the temperature is adjusted back to the initial temperature T_0.

In order to keep the best candidate solution found along the evolution process, elitism was included in the Algorithm 1. It compares the fitness at each iteration and stores the best solution in a side variable S_B, which is returned at the end of the run.

In this work, the stopping condition of the algorithm was set as a maximum number of fitness function evaluations, detailed in Section VII.

IV. Solution Representation: introducing the Geometric Circuit Evolution

The Solution Representation (SR) encodes a candidate solution in a convenient format for the SA to work efficiently. In this work, a SR called Geometric Circuit Evolution (GCE) was developed. The main objects of an electronic circuit E in GCE are: B – the board and $C = \{c_0, c_1, \ldots, c_n\}$ – an array of components.

The board represents a matrix of fixed dimensions $M \times N$. Each position are circuit-equivalent as electrical contacts – pads – insulated from each other, resembling somewhat a rectangular perforated circuit board. A board position can have zero or more component terminals associated with it. The Fig. 1 shows an example of a GCE board filled with some components.

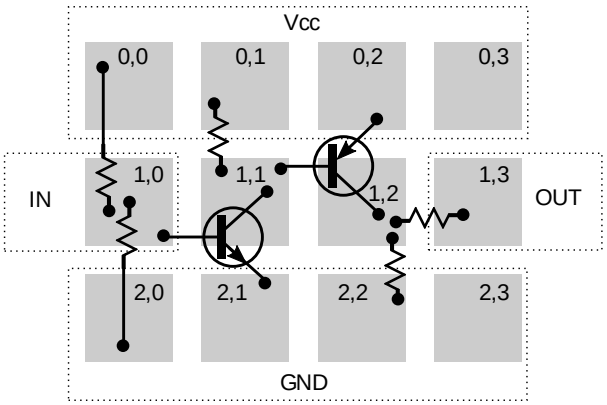


Fig. 1. GCE board example. The gray squares are the board pads and the dotted areas point to the external nodes. Multiple pads inside the same external node are short-circuited.

Algorithm 1 Simulated Annealing algorithm

Require: SR: class that represent a candidate solution (See Section IV). This class must implement the methods *new()*, *mutation()* and *get_fitness()*.
Require: R(): generates a uniform distributed random number $0 \leq x \leq 1$
Input: T_0: Initial temperature
Input: T_r: Reheating threshold
Input: α: Cooling factor. Must be a positive number less than 1, typically very close to 1.

function sann(T_0, T_r, α)
  $T = T_0$
  $S_0 \leftarrow$ SR.new() ▷ Instantiate a new candidate solution
  $S_B \leftarrow S_0$ ▷ Store the best solution (elitism)
  while stopping condition not met **do**
    $S_1 \leftarrow S_0$ ▷ Create a working copy of S_0
    S_1.mutation( )
    $F_0 \leftarrow S_0$.get_fitness( )
    $F_1 \leftarrow S_1$.get_fitness( )
    if $F_0 < F_1$ **then** ▷ Compare the fitness: Less is better
      $S_B \leftarrow S_0$
    else
      $S_B \leftarrow S_1$
    end if
    $\Delta F \leftarrow F_1 - F_0$ ▷ $\Delta F < 0$ means that S_1 is better than S_0
    $A \leftarrow$ false ▷ A: mutation acceptance
    if $\Delta F = 0$ **then**
      $A \leftarrow$ false
    else if $\Delta F < 0$ **then**
      $A \leftarrow$ true
    else
      $p = e^{-\Delta F/T}$
      if R( ) $< p$ **then**
        $A \leftarrow$ true
      end if
    end if
    if A **then**
      $S_0 \leftarrow S_1$
    end if
    $T \leftarrow T \cdot \alpha$
    if $T \leq T_r$ **then** ▷ if below a threshold...
      $T = T_0$ ▷ ...apply reheating
    end if
  end while
  return S_B ▷ return the best solution found
end function

A board, as an object, stores:

- number of components in the circuit (n);
- an array of n component objects;
- $\Omega = \{\omega_{n0}, \ldots, \omega_{nx}, \ldots, \omega_{np}\}$, a set of sets, each containing a fixed node nx attributed to some external pads. For example, the set $\omega_{n0} = \{(2,0),(2,1),(2,2),(2,3)\}$ indicates that those four pads are associated with the external node 0, the GND in the spice simulation. These external nodes are depicted as dotted areas in Fig. 1.

Each component c in GCE stores:

- Its type (for example: resistor, transistor, etc.);
- The position for each of its terminals on the board;
- Its parameters (for example: electrical resistance, model of a BJT transistor, channel width in a FET transistor), stored as an array of variant data type (integer, float or string depending on parameter type).

received 7 July 2022; for review 11 July 2022; accepted 5 Jan 2023

ISSN 2179-1074

Each object in the GCE may have some chosen variables that can be perturbed from the outside. Those variables are called here degrees of freedom (DOF), a correspondence with mechanical systems. When a DOF is perturbed, it calls its own specific generator routine to (possibly) generate a new value and then optionally call a listener routine to act on the value changed. For example, the parameter n of the board (the number of components of the circuit) are a DOF that, when perturbed, generates a uniform distributed random number in a range $[\min, \max]$ and then creates or destroys a number of components to hold exactly n components in memory.

TABLE I. LIST OF DEGREES OF FREEDOM IN GCE

Variable	Object	Type	Description	Possible values	Generator/Listener
n	Board	Structure	The number of components in the circuit	$n \in \mathbb{N}^+, n_{min} \leq n \leq n_{max}$. Configured in each problem.	Uniform distributed random number in the prescribed range. Listener routine: Creates or destroys a number of components to hold exactly n components in memory
c_t	Component	Structure	The type of the component	$c_t \in \{R, Q, Ma\}$, where R is a resistor, Q is a bipolar junction transistor (BJT) and Ma is a MOSFET with fixed channel length of 10 μm	Choose randomly from the prescribed set of discrete values, with specified individual weights. Listener routine: Regenerate all specific component DOFs
cr_{ta}, cr_{tb}	Component c_t=R	Terminal	The position of each of the two terminals of the resistor	$(x, y); 0 \leq x \leq M-1; 0 \leq y \leq N-1$	In the first generation, a uniform distributed random integer within the allowable range is returned. Later generations increment each axis independently by -1, 0 or 1 (random choice)
cr_r	Component c_t=R	Continuous	The resistor electrical resistance in Ω	$1 \leq cr_r \leq 60 \times 10^6$	$cr_r = a \times 10^b$, where a are a uniform random number from the interval $[1, 10]$ and b are a random integer from the interval $[0, 6]$ chosen with weights $[5/135,\ 20/135,\ 30/135,\ 40/135,\ 25/135,\ 10/135,\ 5/135]$
cq_{te}, cq_{tb}, cq_{tc}	Component c_t=Q	Terminal	The position of the emitter, base and collector terminals of the transistor	$(x, y) : 0 \leq x \leq M-1; 0 \leq y \leq N-1$	In the first generation, a uniform distributed random integer within the allowable range is returned. Later generations increment each axis independently by -1, 0 or 1 (random choice)
cq_m	Component c_t=Q	Discrete	The spice model for the transistor	$cq_m \in \{M_0, M_1, \ldots, M_{n-1}\}$, where M are the spice model name (string). Configured in each problem. Example: $cq_m \in \{\text{2N3904}, \text{2N3906}\}$	Choose randomly from the prescribed set of strings, with equal weight
cma_{tg}, cma_{td}, cma_{ts}	Component c_t=Ma	Terminal	The position of the gate, drain and source terminals of the MOSFET	$(x, y); 0 \leq x \leq M-1; 0 \leq y \leq N-1$	In the first generation, a uniform distributed random integer within the allowable range is returned. Later generations increment each axis independently by -1, 0 or 1 (random choice)
cma_m	Component c_t=Ma	Discrete	The spice model for the MOSFET	$cma_m \in \{M_0, M_1, \ldots, M_{n-1}\}$, where M are the spice model name (string). Configured in each problem. Example: $cma_m \in \{\text{NMOS1}, \text{PMOS1}\}$	Choose randomly from the prescribed set of strings, with equal weight
cma_w	Component c_t=Ma	Continuous	Mosfet channel width, in μm	$cma_w \in \mathbb{N}^+, w_{min} \leq n \leq w_{max}$. Configured in each problem.	Uniform distributed random number in the prescribed range

All DOFs have an associated type T, which can be one of the following:

- *Terminal*: represents where a terminal of a component is connected in the circuit. Its change may have great variation in circuit response;
- *Continuous*: defines a continuous variable (e.g. resistance of a resistor). Its change may have smooth impact in circuit response;

received 7 July 2022; for review 11 July 2022; accepted 5 Jan 2023

ISSN 2179-1074

- *Discrete*: defines a discrete variable (e.g. the spice model of a transistor, chosen from a list of options). Its change may imply in moderated variation in circuit response;
- *Structure*: control the structure of the circuit (e.g. the number of components). Its change may cause great variation in circuit response, and can alter the number of DOFs.

Table I lists all DOFs used in GCE, along with its type, their pertaining object, its description, the possible values it can take, generator and listener details.

The main interface between GCE and the SA is the *mutation* procedure, invoked by SA in GCE. No crossover operator was used. The mutation acts in the GCE's DOFs according to the Algorithm 2.

The procedure obtains all the DOFs and visit them one by one. In a visit the DOF may be perturbed (i.e. mutated). Note that structure type DOFs are processed first in a special way, because when perturbed it can create or destroy other DOFs. The mutation occurs with a probability $p_m = p_{mg} \cdot p_s(T)$, where p_{mg} is a general mutation probability and $p_s(T)$ is a specific mutation probability for the DOF type T. Both p_{mg} and $p_s(T)$ are configurable parameters.

The discrimination of $p_s(T)$ for different T allows the mutation to be less frequent for the types of DOFs that, when perturbed, can cause disruptive variations in the circuit response. In this way, the search algorithm can explore the problem space more efficiently.

Algorithm 2 GCE mutation algorithm

Require: get_dofs(): get a current list of all DOFs
Require: R(): generates a uniform distributed random number $0 \leq x \leq 1$
Input: p_{mg}: general mutation probability, $0 \leq p_{mg} \leq 1$
Input: $p_s[T]$: specific mutation probability for the DOF type T, $0 \leq p_s[T] \leq 1, \forall\ T$

```
1:  procedure mutation(p_mg, p_s)
2:      D ← get_dofs()
3:      for all d ∈ D do                          ▷ 1st iteration: only structure DOF
4:          T ← d.type( )                         ▷ get the DOF d type
5:          if T = structure then
6:              if R( ) ≤ p_mg · p_s[T] then
7:                  d.perturb( )
8:                  D ← get_dofs()                ▷ get all DOFs again, because they may have been
                                                    changed
9:              end if
10:         end if
11:     end for
12:     D ← get_dofs()
13:     for all d ∈ D do                          ▷ 2nd iteration: other DOF types
14:         T ← d.type( )
15:         if T ≠ structure then
16:             if R( ) ≤ p_mg · p_s[T] then
17:                 d.perturb( )
18:             end if
19:         end if
20:     end for
21: end procedure
```

A candidate solution creation starts with a board object with its only DOF (n, the number of components, structure type) set to zero. This DOF is perturbed, which causes n to assume a positive random value $n_{min} \leq n \leq n_{max}$. Its listener routine acts on value changed, creating n component objects (each one adding DOFs to the candidate solution). Each component in turn will have its DOFs perturbed, starting by the structure type DOF c_t (component type), which will define the component type and create its specific DOFs. Then, all the remaining components DOFs are perturbed, resulting in a random electronic circuit.

received 7 July 2022; for review 11 July 2022; accepted 5 Jan 2023

ISSN 2179-1074

All the information needed to express it as a spice netlist are contained inside the object's DOFs. This operation are called translation, and are depicted in Algorithm 3. The candidate solution netlist are then sent to the spice simulator in order to get its performance. Section VI presents the details of this step.

Algorithm 3 GCE translation to spice netlist

Require: Each component type must implement the method get_netlist().
Require: START_NODE: integer that states the first node number to use in the internal nodes
Require: get_external_node(P): function that return the external node number (≥ 0) for the board position P or -1 if P does not have an external node number assigned
Input: C: array of component objects

function translation(C)
  $S \leftarrow$ '' ▷ Empty string initialization
  for all c $\in$ C **do**
    $s \leftarrow$ c.get_netlist( )
    $S \leftarrow S +\ s +$ '\n' ▷ Concatenate and add new line
  end for
  return S ▷ The circuit's spice netlist
end function
function Q.get_netlist( ) ▷ Example for BJT
  return 'Q ' + pos2node(cq_{tc}) + ' ' + pos2node(cq_{tb}) + ' ' + pos2node(cq_{te}) + ' ' + cq_m
end function
function pos2node(P) ▷ Get node number from board position $P = (x, y)$
  static $M \leftarrow$ map[] ▷ Create map data type that keep its contents between function calls
  static $c \leftarrow$ START_NODE ▷ Counter that retain memory between function calls
  $e \leftarrow$ get_external_node(P)
  if e ≥ 0 **then**
    $n \leftarrow e$
  else
    if P $\in$ M **then**
      $n \leftarrow M[P]$
    else
      $n \leftarrow c$
      M.add[P]
      $M[P] \leftarrow n$
      $c \leftarrow c + 1$
    end if
  end if
  return n ▷ return the node number
end function

V. Problem formulation

The circuit to be synthesized in SANN-GCE must be prescribed through a problem formulation, which involves the following steps:

- Defining a test fixture (TF), that is a fixed circuit that supply power and tests the circuit that will be created – called here evolved circuit (EC). The interface between TF and EC are the external nodes;
- Specifying the manipulated variable(s) (MV). This is the TF circuit parameters that are changed in order to test the EC;
- Prescribing the responding variable(s) (RV). This is usually voltage or current in TF node(s) that indicates the EC response to the stimulus given by the MVs;

received 7 July 2022; for review 11 July 2022; accepted 5 Jan 2023

ISSN 2179-1074

- Writing the spice script that drives the simulation, based on the previous items of this list. The script must contain the type of analysis to run and the variables to be saved for the fitness calculation. It may contain component models and simulation options;
- Writing the fitness calculation function. Its input is the spice simulation output, normally a collection of arrays. This will be compared with a desired reference output, giving a real number representing a fitness of the EC. The lower the fitness, the better the circuit. This is the primary metric, the only that drives the meta-heuristic optimization;
- Writing secondary metric functions. This is optional, and is also calculated comparing the spice simulation output with a desired reference, but using other formulas. This is just to be logged and is not used in the meta-heuristic optimization.

VI. Circuit simulation

The candidate solutions are assessed in order to get some performance metrics of each one. The main metric that drives the evolutionary process is fitness f, which the smaller the better. Thus, the objective of the meta-heuristic is to optimize the circuit, minimizing f. The calculation of fitness is obtained through the following steps:

1) Obtain the raw spice netlist S' from the candidate solution E, using the Algorithm 3;
2) Pre-process S'. This comprises: sequentially numbering the components and removing components that have all their terminals short-circuited. The result are saved in S;
3) Run spice simulator with netlist S as input. The result are θ, a data structure that contains vectors for the voltages and/or currents requested for some or all circuit nodes over time, frequency or temperature, depending on the type of spice analysis requested;
4) Process θ to obtain f and some other metrics, using the problem's objective function $P()$.

The spice simulator used in this work was the Ngspice [19], an open source and multi platform circuit simulator compatible with PSPICE and LTSPICE.

VII. Test circuits

In order to test the SANN-GCE, it was selected seven problems (test circuits) as use cases for the algorithm. These test circuits are the same as in [10] and [12], because they presented seven different use cases with detailed results, and all tests were run 50 times, which allowed us to make a robust comparison.

The test circuits was divided in two sets: 1) Non-computational circuits, which comprises a temperature sensor circuit, a Gaussian function and a voltage reference; 2) Computational circuits: squaring, square root, cubing, and cube root.

Table IV shows the SANN-GCE parameters used in each test circuit. These parameters were chosen after preliminary runs with different values for each parameter. The best combination found experimentally are used in each test circuit. The experimental nature of the parameter tuning is typical in metaheuristics algorithms [20]. Fine-tuning each parameter for each test circuit was outside the scope of this work.

Each candidate solution is accessed to get its primary and secondary metrics. Equation (2) calculates the fitness value f, the primary metric that drives the optimization algorithm.

received 7 July 2022; for review 11 July 2022; accepted 5 Jan 2023

ISSN 2179-1074

$$f = \sum_{i=0}^{N_p-1} w_i e_i \tag{2}$$

where N_p is the number of points in the result of the spice simulation, and e_i is the absolute error between the circuit responding (output) variable X_i and the desired (reference) value $\tilde{X}_i$, calculated as follows:

$$e_i = |X_i - \tilde{X}_i| \tag{3}$$

The w_i is a weight factor that increases the penalty to the fitness f when fitting points has an error above a predefined threshold X_{th}:

$$w_i = \begin{cases} w_b & e_i \leq X_{th} \\ 10 \cdot w_b & \text{otherwise} \end{cases} \tag{4}$$

The w_b is a base weight that can be configured in each test circuit. Other metric calculated is the Mean Average Error (MAE), a secondary one, obtained as the following:

$$MAE = \frac{1}{N_p} \sum_{i=0}^{N_p-1} e_i \tag{5}$$

Other secondary metric was the hits %. This calculates the percentage of fitting points that has an error below the threshold X_{th}:

$$hits_{\%} = \frac{1}{N_p} \sum_{i=0}^{N_p-1} [e_i \leq X_{th}] \tag{6}$$

Table II presents the values of some of the variables referenced in the above equations for each test circuit and the following Sections describe each test circuit in detail.

TABLE II. PARAMETERS FOR CALCULATING THE METRICS FOR TEST CIRCUITS

Circuit	N_p	X_{th}	w_b
Temperature sensor	21	0.1	1
Gaussian function	101	5×10^{-9}	10^6
Voltage reference	105	0.02	1
Squaring	21	0.05×0.25^2	1
Square root	21	$0.05 \times \sqrt{0.5}$	1
Cubing	21	0.05×0.25^3	1
Cube root	21	$0.05 \times \sqrt[3]{0.25}$	1

A. Temperature sensor

The temperature sensor problem aims to automatically design an analog electronic circuit that generates an output voltage proportional to the circuit temperature. See Fig. 2 for the test fixture used in this problem. The spice simulator changes the circuit temperature following the sequence $T = [0, 5, 10, \ldots, 100]$, resulting in $N_p = 21$ simulation points. The circuit output is the voltage in

received 7 July 2022; for review 11 July 2022; accepted 5 Jan 2023

ISSN 2179-1074

node #3, denoted as X_i, where i is the corresponding point in the input vector T. The desired output is given by $\tilde{X}_i = T_i/10$.

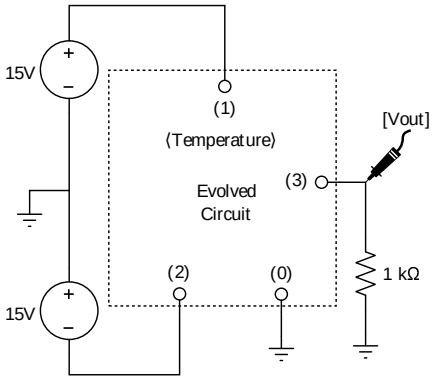


Fig. 2. Test fixture for the temperature sensor circuit. External node numbers are between parenthesis. Manipulated variables are between angle brackets and responding variables are between square brackets.

B. Gaussian function

The Gaussian function circuit tries to make its output current approximate a Gaussian curve as a function of the input voltage. The Fig. 3 presents the test fixture used in this circuit. The input source change its voltage linearly as $Vin = [2, 2.01, 2.02, \ldots, 3]$, resulting in $N_p = 101$ simulation points. The output X_i is the current flowing out to node #3. The desired output for the i-th input is given by the Equation (7), where $a = 80 \times 10^{-9}$ is the peak value of the function, $b = 2.5$ is the distribution average value and $c = 0.1$ is the standard deviation.

$$\tilde{X}_i = a \exp\left(- \frac{(Vin_i - b)^2}{2c^2} \right) \tag{7}$$

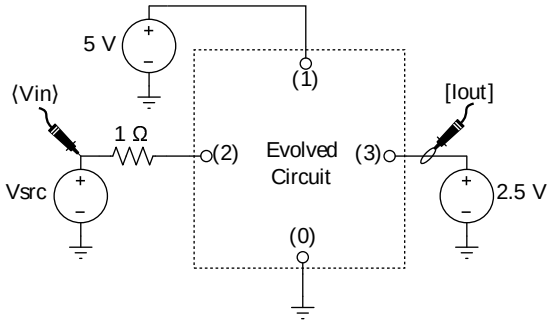


Fig. 3. Test fixture for the Gaussian function circuit. External node numbers are between parenthesis. Manipulated variables are between angle brackets and responding variables are between square brackets.

C. Voltage reference

The voltage reference circuit tries to keep its output voltage constant when subjected to variations in supply voltage and circuit temperature. Its test fixture is depicted in Fig. 4. The temperature changes according to $T = [0, 25, 50, 75, 100]$ and for each temperature, the supply voltage changes following the sequence $Vin = [4, 4.1, 4.2, \ldots, 6]$, thus totaling $5 \times 21 = 105$ simulation points. The output X_i is

received 7 July 2022; for review 11 July 2022; accepted 5 Jan 2023

ISSN 2179-1074

the voltage of node #2, referenced to ground. The desired output $\tilde{X}_i$ is 2 volts for all the 105 simulation points, whatever the input.

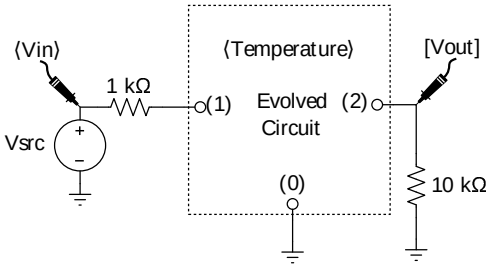


Fig. 4. Test fixture for the voltage reference circuit. External node numbers are between parenthesis. Manipulated variables are between angle brackets and responding variables are between square brackets.

D. Computational circuits

The four computational circuits tries to approximate a particular mathematical curve. Fig. 5 shows the test fixture used in this class of problem. The input source Vin changes its voltage in a linear ramp sampled in $N_p = 21$ points according to the limits given in Table III. The output X_i is the voltage of node #4. The desired output voltage $\tilde{X}_i$ is also given in the aforementioned table.

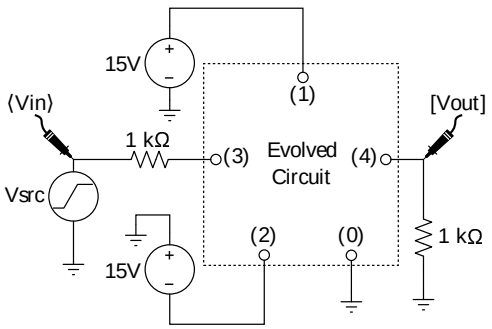


Fig. 5. Test fixture for the four computational circuits (squaring, square root, cubing, and cube root). External node numbers are between parenthesis. Manipulated variables are between angle brackets and responding variables are between square brackets.

TABLE III. INPUT VOLTAGE RANGE AND THE DESIRED OUTPUT VOLTAGE FOR EACH OF THE COMPUTATIONAL CIRCUITS

Circuit	Vin range	$\tilde{X}_i$
Squaring	$[-0.25, 0.25]$	Vin^2
Square root	$[0, 0.5]$	$\sqrt{Vin}$
Cubing	$[-0.25, 0.25]$	Vin^3
Cube root	$[-0.25, 0.25]$	$\sqrt[3]{Vin}$

VIII. RESULTS AND DISCUSSION

Each test circuit described in Section VII was executed 50 times in SANN-GCE algorithm, using parameters given in Table IV. The runs were set to stop when they reach 3×10^6 fitness function evaluations, a value equal to the number of times a spice simulation is called.

received 7 July 2022; for review 11 July 2022; accepted 5 Jan 2023

ISSN 2179-1074

The choice of this stopping criterion was made to make the comparison between different algorithms fairer, as the simulation is the most time-consuming step in each algorithm cycle, and the number of cycles (or generations, in population-based meta-heuristics) may not correctly indicate the amount of work performed by the spice simulator.

The results of the ACID-GE [10] and ACID-MGE [12] algorithms were used to compare with those in this work, with a focus in the ACID-MGE that presents a better performance. Data from the results of these two algorithms were obtained from tables 4, 5 and 7 of [12].

Figs. 7 and 8 shows the performance curves for the best circuit found in each problem using the SANN-GCE. Each plot contains a reference and the actual circuit response. Table V shows the comparison for results of SANN-GCE, ACID-MGE and ACID-GE, in various metrics. The meaning of the columns in the table are:

- SR%: Success Ratio: Relative number of successful runs. A successful run is one that achieves 100 % hits;
- BF: Best Fitness, calculated for the best solution found in each run, according Equation (2);
- MAE: Mean Average Error, calculated according Equation (5);
- Hits %: Percentage of fitting points with an error below a threshold, calculated according Equation (6);
- #Gen success: Number of generations where success (100 % hits) was achieved. In SANN-GCE this is not a directly counted value, but an equivalent value (because it does not use generations like those based on genetic algorithms). This equivalent number of generations is the count of fitness function evaluations divided by the population count in the compared algorithm. This divisor is 1000 in the referred ACID-MGE and ACID-GE;
- NCBC: Number of Components for the Best Circuit. It counts only the components in the evolved circuit, not those from the test fixture.
- Runtime: Normalized time to complete 50 runs. ACID-MGE used a cluster of 8 computers, ACID-GE used a cluster of 5 computers and SANN-GCE used one computer. To allow for a reasonable comparison of run-times between these different algorithms, we performed the following normalization: in ACID-GE and ACID-MGE the time shown are an estimate to execute 50 runs of 3000 generations each in just one computer. In SANN-GCE, the time shown was the actual time to execute 50 runs, each run stopped after 3 million evaluations of the fitness function (equivalent to 3000 generations of a population with 1000 candidate solutions, as used by ACID-MGE and ACID-GE).

The data for MAE in the non-computing circuits are not available for the ACID-GE and ACID-MGE, as the source article does not report these values. Also, $\mathrm{MAE}_{\min}$ is not available for ACID-GE in all test circuits.

All metrics shown in Table V are less is better, except those marked with an asterisk in the header (SR% and $\mathrm{Hits}_{\mathrm{mean}}$), where greater values mean better results. With that, in order to ease the comparison of a metric M between SANN-GCE and another algorithm α, we define $R_{M,\alpha}$ – the SANN-GCE performance ratio – as the following:

Brazilian Microwave and Optoelectronics Society-SBMO *received 7 July 2022; for review 11 July 2022; accepted 5 Jan 2023*
Brazilian Society of Electromagnetism-SBMag *ISSN 2179-1074*

$$R_{M,\alpha} = \begin{cases} \dfrac{M_{\text{SANN-GCE}}}{M_\alpha} & \text{if bigger } M \text{ means better, or} \\ \dfrac{M_\alpha}{M_{\text{SANN-GCE}}} & \text{if smaller } M \text{ means better} \end{cases} \tag{8}$$

Therefore, $R < 1$ and $R > 1$ means, respectively, a worse and a better result for SANN-GCE, regardless of whether a particular metric follows the logic of smaller is better or greater is better.

The SANN-GCE performance ratio regarding ACID-MGE was calculated individually for each metric in all test circuits. Then these data were grouped by metric and shown in Fig. 6 as a boxplot, to facilitate comparison between SANN-GCE and ACID-MGE.

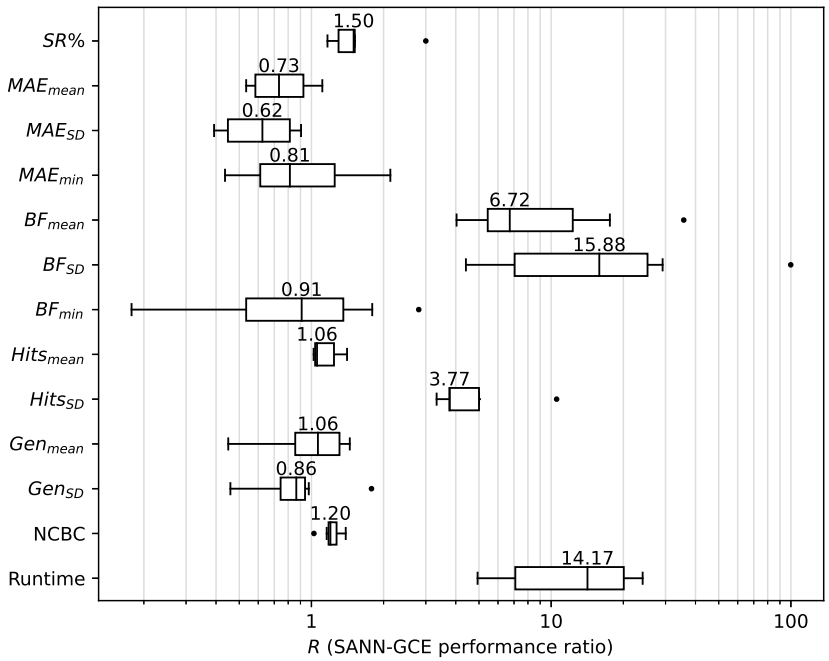


Fig. 6. Performance ratio between SANN-GCE and ACID-MGE in each metric. The statistics for each metric was calculated over all test circuits. Values > 1 denotes SANN-GCE better. The numbers above each box are the median.

It can be seen in Table V that SANN-GCE have a consistent performance advantage in various metrics, such as in SR%, BF (mean and standard deviation), hits (mean and standard deviation) which are better in 100% of test circuits when compared to both ACID-MGE and ACID-GE.

Fig. 6 presents that, compared to ACID-MGE, SANN-GCE has a median performance ratio for SR% equal to 1.5, calculated over all the test circuits. For $\mathrm{BF}_{\mathrm{mean}}$ it was 6.72. In $\mathrm{BF}_{\mathrm{SD}}$ the median was 15.88. $\mathrm{Hits}_{\mathrm{mean}}$ and $\mathrm{Gen}_{\mathrm{mean}}$ was both 1.06, $\mathrm{Hits}_{\mathrm{SD}}$ was 3.77. NCBC was 1.2 and Runtime was 14.17.

Continuing with the same analysis method described above, SANN-GCE was worse than ACID-MGE in the MAE (mean, standard deviation, min), $\mathrm{BF}_{\mathrm{min}}$ and $\mathrm{Gen}_{\mathrm{SD}}$ metrics, with a median of R equal to 0.73, 0.62, 0.81, 0.91, 0.86, respectively.

These results shown that the SANN-GCE have a notable performance, specially lower variability between runs ($\mathrm{BF}_{\mathrm{SD}}$) and also good mean value ($\mathrm{BF}_{\mathrm{mean}}$) for the primary metric.

The runtime also stands out, with a median 14.17 times faster than the ACID-MGE. This result must be taken with care due to the inherent differences of hardware and software, but we can say that the

simplicity of Simulated Annealing played an important role in this regard. As stated before, it uses only the mutation operation to modify a candidate solution, unlike others algorithms that also uses the crossover operator, such as ACID-GE/MGE which are based on genetic algorithms.

We attribute the two favorable results in the primary metric to the slow and incremental nature of the SANN-GCE algorithm for using mutation only, compared to the more disruptive crossover operator used in ACID-MGE. This may also explain the slightly less favorable $\mathrm{BF}_{\mathrm{min}}$ obtained in our algorithm: the higher level of search space exploration done in ACID-MGE by its crossover operator might be able to more easily achieve a better fitness during optimization.

IX. Conclusion

In this paper we presented GCE, a new solution representation for simulation based analog electronic circuits, working together with simulated annealing, a classic meta-heuristic for global optimization. The GCE internally represents a candidate solution as a collection of objects, with components placed geometrically on a board similar to a physical perforated board.

It was demonstrated that our algorithm can be used to successfully design analog electronic circuits automatically using a mutation-only approach, using only one candidate solution in each generation.

Using only the mutation operator made the algorithm simpler and resulted in low resource usage, as it operates on one candidate solution at a time. This may explain the fact that the median runtime was 14.17 times faster on SANN-GCE when compared to ACID-MGE.

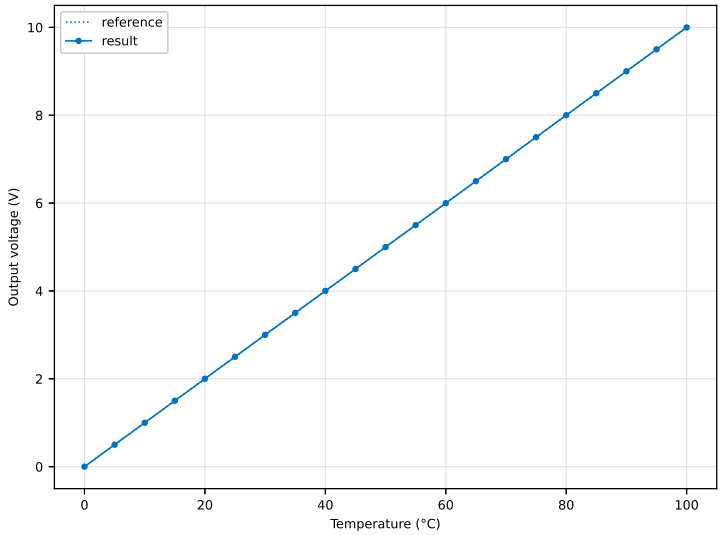
SANN-GCE was tested with seven test circuits, and for each one, 13 metrics was collected and discussed. The results showed good performance in various metrics, specially consistent performance between runs, showing lower standard deviation than the others algorithms compared (e.g. in median, SANN-GCE have the standard deviation of BF $15.88\times$ lower than ACID-MGE. For Hits it was $3.77\times$ lower). In real use this means that we could get a circuit with reasonable performance with fewer runs. Also, for metrics SR%, BF (mean and standard deviation) and hits (mean and standard deviation), SANN-GCE was better in all circuits tested.

The algorithm parameters used in this work were chosen experimentally after few trials and was not investigated in details. Their influence on algorithm performance needs further investigation and is planned as a future work.

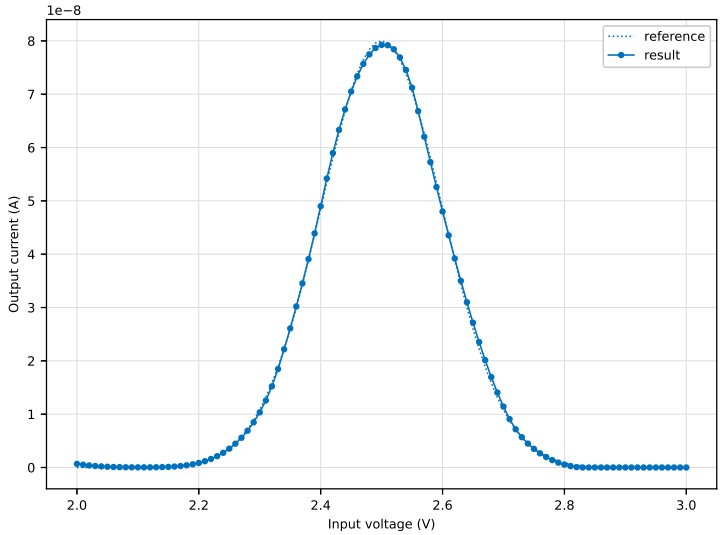
Appendix 1 - Parameters and results

In this appendix we show Fig. 7 and Fig.8 with the performance curves of the best circuit found in each problem. We also present Table IV and Table V, one showing the parameters used in the test circuits and the other containing the results.

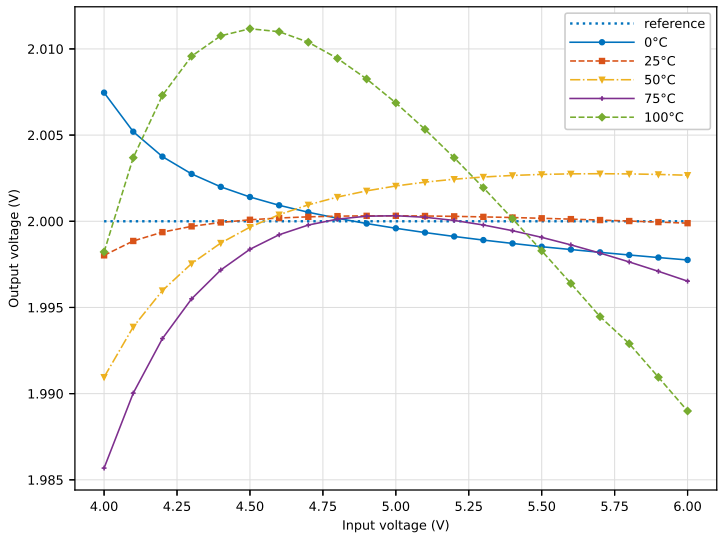
Brazilian Microwave and Optoelectronics Society-SBMO *received 7 July 2022; for review 11 July 2022; accepted 5 Jan 2023*
Brazilian Society of Electromagnetism-SBMag *ISSN 2179-1074*



(a)



(b)



(c)

Fig. 7. Performance of the best non-computing circuits obtained with the SANN-GCE algorithm. (a) Temperature sensor, (b) Gaussian function, (c) Voltage reference.

DOI: http://dx.doi.org/10.1590/2179-10742023v22i1265775

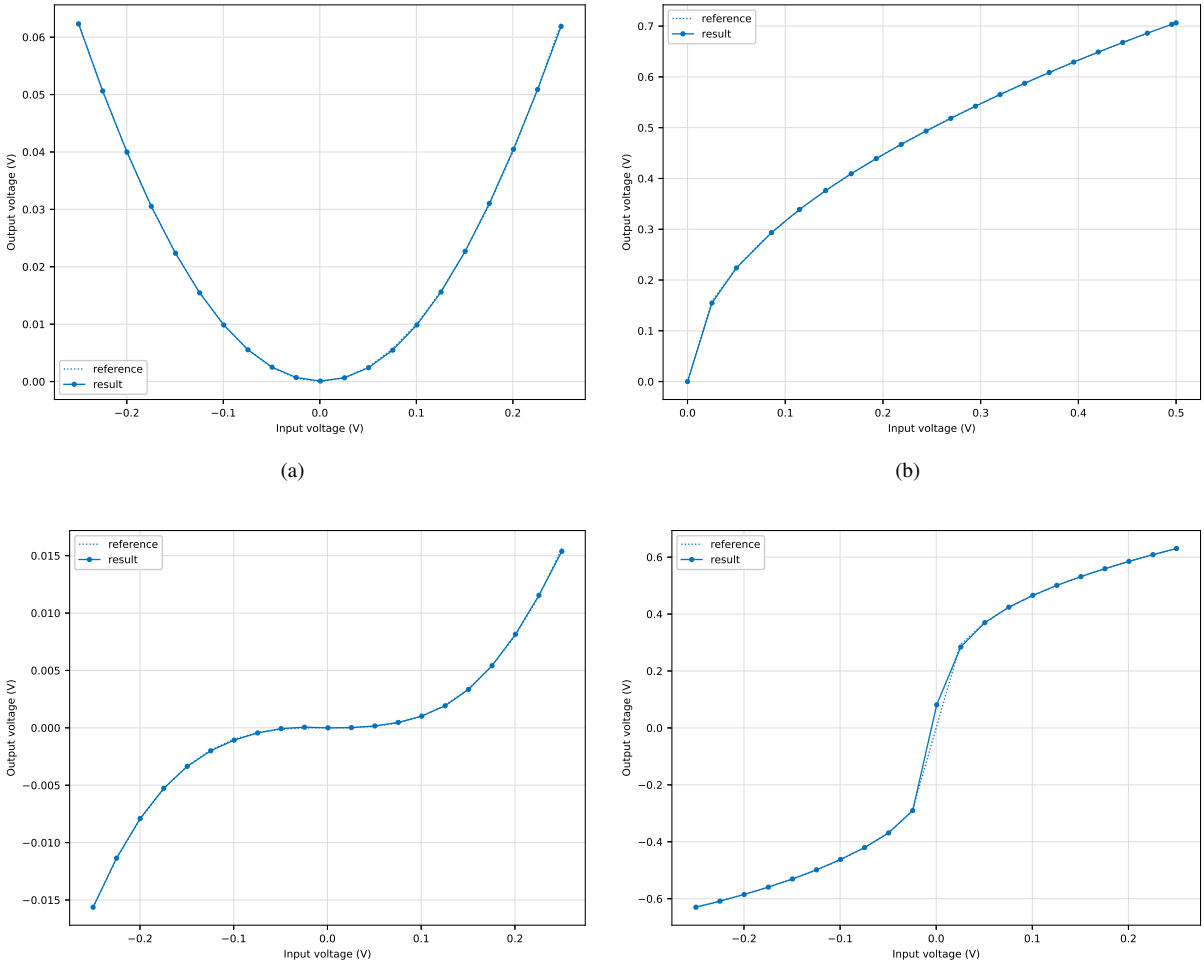


Fig. 8. Performance of the best computational circuits obtained with the SANN-GCE algorithm. (a) Squaring, (b) Square root, (c) Cubing, (d) Cube root.

received 7 July 2022; for review 11 July 2022; accepted 5 Jan 2023

ISSN 2179-1074

TABLE IV. PARAMETERS USED IN THE TEST CIRCUITS

Parameter	Test circuit						
	Temp. sensor	Gaussian function	Voltage reference	Squaring	Square root	Cubing	Cube root
Stopping condition	3×10^6 fitness function evaluations (1:1 relation with number of spice simulations)						
T_0	100						
T_r	1×10^{-9}						
α	0.9999						
p_{mg}	0.015						
$p_s[T]$	Structure $\mapsto 0.25$; Others $\mapsto 1.0$						
Test fixture	See Figs. 2 to 5						
n_{min}, n_{max}	$7, 42$						
$M \times N$	4×4	4×4	4×4	4×4	4×4	5×5	6×6
External nodes map	$(0,1) \mapsto 0$ $(0,2) \mapsto 1$ $(2,3) \mapsto 3$ $(3,1) \mapsto 2$ $(3,2) \mapsto 0$	$(0,1) \mapsto 1$ $(0,2) \mapsto 1$ $(1,0) \mapsto 2$ $(2,3) \mapsto 3$ $(3,1) \mapsto 0$ $(3,2) \mapsto 0$	$(1,0) \mapsto 1$ $(1,3) \mapsto 2$ $(3,1) \mapsto 0$ $(3,2) \mapsto 0$	$(0,1) \mapsto 0$ $(0,2) \mapsto 1$ $(1,0) \mapsto 3$ $(2,3) \mapsto 4$ $(3,1) \mapsto 2$ $(3,2) \mapsto 0$	$(0,1) \mapsto 0$ $(0,2) \mapsto 1$ $(1,0) \mapsto 3$ $(2,3) \mapsto 4$ $(3,1) \mapsto 2$ $(3,2) \mapsto 0$	$(0,1) \mapsto 0$ $(0,2) \mapsto 1$ $(2,0) \mapsto 3$ $(2,4) \mapsto 4$ $(4,2) \mapsto 2$ $(4,3) \mapsto 0$	$(0,2) \mapsto 0$ $(0,3) \mapsto 1$ $(2,0) \mapsto 3$ $(3,5) \mapsto 4$ $(5,2) \mapsto 2$ $(5,3) \mapsto 0$
Component types and weights	R $\mapsto 70\%$ Q $\mapsto 30\%$	R $\mapsto 50\%$ Ma $\mapsto 50\%$	R $\mapsto 70\%$ Q $\mapsto 30\%$	R $\mapsto 70\%$ Q $\mapsto 30\%$	R $\mapsto 70\%$ Q $\mapsto 30\%$	R $\mapsto 70\%$ Q $\mapsto 30\%$	R $\mapsto 70\%$ Q $\mapsto 30\%$
cq_m	{2N3904, 2N3906}	N.A.	{2N3904, 2N3906}	{2N3904, 2N3906}	{2N3904, 2N3906}	{2N3904, 2N3906}	{2N3904, 2N3906}
cma_m	N.A.	{NMOS, PMOS}	N.A.	N.A.	N.A.	N.A.	N.A.
w_{min}, w_{max}	N.A.	10, 199	N.A.	N.A.	N.A.	N.A.	N.A.

Brazilian Microwave and Optoelectronics Society-SBMO *received 7 July 2022; for review 11 July 2022; accepted 5 Jan 2023*
Brazilian Society of Electromagnetism-SBMag

ISSN 2179-1074

TABLE V. RESULTS OF THE SANN-GCE ALGORITHM COMPARED TO ACID-MGE AND ACID-GE. LESS IS BETTER FOR ALL METRICS, EXCEPT THOSE MARKED WITH AN ASTERISK (*). THE BEST RESULTS FOR A METRIC IN EACH TEST CIRCUIT ARE IN BOLD.

PART 1:

Test Circuit	Algorithm	SR (%)*	MAE (mV)			NCBC	Runtime
			Mean	± SD	min		
Temperature Sensor	ACID-MGE	70.0	NA	NA	NA	42	433h
	ACID-GE	42.0	NA	NA	NA	**28**	312h
	SANN-GCE	**100.0**	14.4	5.4	0.6	41	**17h59m**
Gaussian function	ACID-MGE	58.0	NA	NA	NA	37	433h
	ACID-GE	24.0	NA	NA	NA	41	312h
	SANN-GCE	**88.0**	9.53E-07	5.91E-07	3.00E-07	**29**	**18h54m**
Voltage reference	ACID-MGE	8.0	NA	NA	NA	42	433h
	ACID-GE	8.0	NA	NA	NA	**32**	312h
	SANN-GCE	**12.0**	11.0	3.7	2.8	35	**87h47m**
Squaring	ACID-MGE	84.0	**0.53**	**0.24**	**0.08**	42	433h
	ACID-GE	52.0	0.93	0.41	NA	**28**	312h
	SANN-GCE	**98.0**	0.61	0.31	0.12	35	**25h05m**
Square root	ACID-MGE	66.0	4.01	**2.47**	**0.23**	44	433h
	ACID-GE	44.0	5.55	4.32	NA	**35**	312h
	SANN-GCE	**100.0**	**3.61**	2.73	0.53	38	**30h33m**
Cubing	ACID-MGE	84.0	**0.10**	**0.04**	0.05	47	433h
	ACID-GE	76.0	0.18	0.07	NA	40	312h
	SANN-GCE	**98.0**	0.19	0.10	0.05	**37**	**46h52m**
Cube root	ACID-MGE	22.0	**7.13**	3.70	2.04	57	433h
	ACID-GE	2.0	10.81	**0.00**	NA	41	312h
	SANN-GCE	**66.0**	11.88	7.92	**0.96**	41	**87h45m**

PART 2:

Test Circuit	Algorithm	BF			Hits (%)		#Gen success	
		Mean	± SD	min	Mean*	± SD	Mean	± SD
Temperature Sensor	ACID-MGE	2.13	3.34	0.033	97.1	5.3	711.2	726.2
	ACID-GE	5.29	6.53	0.169	93.0	7.9	1278.9	884.6
	SANN-GCE	**0.30**	**0.11**	**0.012**	**100.0**	**0.0**	**492.0**	**407.8**
Gaussian function	ACID-MGE	1.00	1.75	**0.028**	94.1	9.8	1023.2	719.6
	ACID-GE	6.70	6.78	0.040	77.3	17.6	1367.4	**688.5**
	SANN-GCE	**0.15**	**0.26**	0.030	**99.3**	**2.6**	**961.4**	832.1
Voltage reference	ACID-MGE	19.35	14.04	**0.053**	64.2	23.3	**1205.0**	**863.3**
	ACID-GE	26.57	16.23	0.112	53.8	25.0	1581.0	917.0
	SANN-GCE	**4.63**	**3.18**	0.298	**90.4**	**7.0**	1583.3	1018.9
Squaring	ACID-MGE	0.06	0.14	**0.002**	97.7	6.5	**671.5**	**574.9**
	ACID-GE	0.73	1.17	0.009	81.0	28.5	1176.9	828.3
	SANN-GCE	**0.015**	**0.019**	0.003	**99.8**	**1.3**	706.1	634.5
Square root	ACID-MGE	2.71	5.78	**0.003**	89.0	24.2	1154.9	**655.0**
	ACID-GE	6.25	7.61	0.028	74.0	32.3	1329.3	797.3
	SANN-GCE	**0.08**	**0.06**	0.011	**100.0**	**0.0**	**816.0**	671.8
Cubing	ACID-MGE	0.03	0.08	0.001	95.7	13.7	**539.8**	**361.7**
	ACID-GE	0.05	0.12	0.002	92.2	19.3	1008.0	703.6
	SANN-GCE	**0.004**	**0.005**	0.001	**99.8**	**1.3**	1200.0	787.7
Cube root	ACID-MGE	13.87	25.02	0.036	70.2	29.4	1561.1	587.8
	ACID-GE	76.95	23.74	0.227	11.5	20.2	1885.0	**0.0**
	SANN-GCE	**0.79**	**1.17**	**0.020**	**95.5**	**7.8**	**1297.0**	918.3

Brazilian Microwave and Optoelectronics Society-SBMO *received 7 July 2022; for review 11 July 2022; accepted 5 Jan 2023*
Brazilian Society of Electromagnetism-SBMag

ISSN 2179-1074

ACKNOWLEDGMENTS

This study was financed in part by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* – Brasil (CAPES) – Finance Code 001.

REFERENCES

[1] S. E. Sorkhabi and L. Zhang, "Automated topology synthesis of analog and RF integrated circuits: A survey," *Integration - the VLSI Journal*, vol. 56, pp. 128–138, JAN 2017.

[2] R. Harjani, R. Rutenbar, and L. Carley, "OASYS: a framework for analog circuit synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 12, pp. 1247–1266, 1989.

[3] H. Koh, C. Sequin, and P. Gray, "OPASYN: a compiler for CMOS operational amplifiers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 2, pp. 113–125, 1990.

[4] F. Jiao and A. Doboli, "Three learning methods for reasoning-based synthesis of novel analog circuits," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, p. 2411–2414, 2016.

[5] M. Meissner and L. Hedrich, "FEATS: Framework for explorative analog topology synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 2, pp. 213 – 226, 2015.

[6] J. Koza, F. Bennett III, D. Andre, and M. Keane, "Synthesis of topology and sizing of analog electrical circuits by means of genetic programming," *Computer Methods in Applied Mechanics and Engineering*, vol. 186, no. 2, pp. 459–482, 2000.

[7] J. R. Koza, D. Andre, M. A. Keane, and F. H. Bennett III, *Genetic programming III: Darwinian invention and problem solving*. San Francisco: Morgan Kaufmann, 1999, vol. 3.

[8] C. Mattiussi and D. Floreano, "Analog genetic encoding for the evolution of circuits and networks," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 5, pp. 596–607, 2007.

[9] T. McConaghy, P. Palmers, M. Steyaert, and G. G. E. Gielen, "Trustworthy genetic programming-based synthesis of analog circuit topologies using hierarchical domain-specific building blocks," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 4, pp. 557–570, 2011.

[10] F. Castejón and E. J. Carmona, "Automatic design of analog electronic circuits using grammatical evolution," *Applied Soft Computing*, vol. 62, pp. 1003–1018, jan 2018.

[11] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 349–358, Aug 2001.

[12] F. Castejón and E. J. Carmona, "Introducing modularity and homology in grammatical evolution to address the analog electronic circuit design problem," *IEEE Access*, vol. 8, pp. 137 275–137 292, aug 2020.

[13] Z. Rojec, A. Burmen, and I. Fajfar, "Analog circuit topology synthesis by means of evolutionary computation," *Engineering Applications of Artificial Intelligence*, vol. 80, pp. 48–65, APR 2019.

[14] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983. [Online]. Available: http://www.jstor.org/stable/1690046

[15] D. Delahaye, S. Chaimatanan, and M. Mongeau, *Simulated Annealing: From Basics to Applications*. Cham: Springer International Publishing, 2019, pp. 1–35.

[16] P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated annealing*. Dordrecht: Springer Netherlands, 1987, pp. 7–15.

[17] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[18] D. T. Pham and D. Karaboga, *Intelligent optimisation techniques : genetic algorithms, tabu search, simulated annealing and neural networks*. Springer London, 2000.

[19] Ngspice. (2021) Ngspice, the open source spice circuit simulator. [Online]. Available: http://ngspice.sourceforge.net/

[20] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Springer Berlin Heidelberg, 2003.

  ISSN 2179-1074