electronics

MDPI

Article

Design and Parameterization of Analog Electronic Circuits Using Stochastic Optimization Algorithms: A Comparative Study with BJT, FET, and OECT Devices

Stefan Bosse * and Sanjeev Kumar

Practical Computer Science Group, Department of Computer Science, University of Koblenz, 56070 Koblenz, Germany; skumar@uni-koblenz.de

* Correspondence: sbosse@uni-koblenz.de

Abstract

Despite the rise and wide spread of digital technology, analog circuits are experiencing a resurgence for specific applications due to their efficiency in processing physical signals with fewer transistors and lower energy requirements. They offer advantages in explainability and stability, particularly in sensor systems. However, designing these circuits poses challenges, particularly in optimizing them against issues like component aging and environmental dependence. Current research aims to enhance automation in analog circuit design using optimization methods, including stochastic algorithms like Genetic Algorithms (GA), Simulated Annealing (SA), and Particle Swarm Optimization (PSO). These methods improve convergence and fault tolerance, while dealing with the high nonlinearity of analog circuits with transistors. The focus of this paper is on solving multivariate optimization problems with multiple objective functions for analog circuits. We evaluate and compare the component optimization of a small-signal amplifier, operational amplifier, and a sigmoid function circuit using gradient-based and stochastic methods for bipolar junction, field-effect, and organic electro-chemical transistors, differing in transistor transconductance and transfer curves. We show that PSO outperforms all other optimization algorithms in our experiments for all considered circuit classes, showing the highest convergence rates between 40–100% (depending on the circuit class) and the lowest optimization times. The OECT components were manufactured with a drop-on-demand ink-jet process and experimentally characterized, finally deriving an electronic functional digital twin model from measuring data. This is a simulation study but uses data and models from real devices.

Keywords: analog circuits; analog computing; electronic simulation; genetic algorithms; stochastic algorithms; optimization problems; automated design; organic electronics

Check for updates

Academic Editor:
Esteban Tlelo-Cuautle

Received: 10 August 2026
Revised: 18 October 2026
Accepted: 20 October 2026
Published: 22 October 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Despite the dominance of digital technology, there is currently a return to analog circuit principles for specific applications. The processing of physical signals requires electronic circuits to detect or generate signals, and there is still a need for analog circuits. Analog circuits can perform a series of functional calculations approximated with acceptable loss of accuracy with a significantly reduced number of transistors and, therefore, generally have a reduced energy and area requirement. Functional calculations, e.g., computing the sigmoid function, are an integral part of sensor signal processing, e.g., sensor calibration

and fusion. Analog circuits can also contribute to improved explainability and validity compared to (software-based) digital circuits, since analog calculation circuits can represent an image of mathematical models (dualism and bijectivity), and in particular, can have increased stability to disturbances in the energy supply (essentially in sensor nodes supplied autonomously via local energy generation).

The design of analog electronic circuits is a complex and difficult design problem, and is an optimization problem with an optional Pareto solution space. Compared to its digital counterpart, analog design automation is still in its infancy. Generative AI methods are being used for the circuit-level optimization of analog (integrated) circuits, e.g., as described by Zadeh et al. [1], mainly focusing on speed, gain, voltage swing, power, impedance, and frequency bandwidth. This is where this work should start: solving existing problems automatically and researching suitable optimization methods, while avoiding any black-box model approach, such as using Large Language Models (LLMs) or any kind of Deep Learning (DL), but often related to integrated circuit design, as summarized and discussed by Mina et al. [2] and Dutta et al. [3]. In this work, we want to focus on the design of discrete analog circuits not using DL to avoid black-box behavior. The problem of the automatic synthesis of circuits from mathematical model templates or functional models, e.g., multivariate polynomials for sensor calibration and cross-sensitivity elimination, or, more simply, a small AC signal amplifier, is far from being solved. One challenge is the inclusion of increased component parameter scattering, drift, accelerated aging, dependence on environmental parameters, and increased probability of failure, which must be carried out already during the design of the circuits with optimization algorithms, including stochastic algorithms, such as genetic programming combined with electronic simulation, in order to enable more reliable and stable analog calculation circuits for bio-compatible sensor systems. By way of example and frequently necessary, sensor fusion is mentioned in order to correct the cross-sensitivity of sensors with functional calculations.

In analog circuit synthesis, the optimization goals are the minimum number of components (transistors), the accuracy of the calculation (or transfer function to be implemented), stability to faults and failure of individual components, and the minimum energy requirement to enable autonomous power supply. Since these goals are partly contradictory, at best, a Pareto-optimal solution should be found. In this work, we focus on fixed-topology problems, and the associated optimization problems should primarily be solved automatically. Besides silicon-based electronics, organic electronics (OEs) are a key enabling technology for flexible, e.g., bio-compatible, and rapid-prototyping electronics, as described by Granelli et al. [4]. But circuit design with OEs is still a challenge, which should be addressed by our work, too.

Generally, a multivariate optimization problem with multiple target functions has to be solved in the design and parameterization of the analog circuit. An example would be a small-signal amplifier where the static operating point, the signal distortion, and the gain have to be optimized. Both the structure of the circuit and the parameterization of the components can be optimized so that a given mathematical model can be approximated using this analog circuit within predetermined limits. In many cases, a suitable circuit topology with a minimum number of components is already known, but the computation of the component values can be challenging, especially with the addition of component variation. The solution of a multivariate and multi-objective optimization problem using analytical and numerical methods with highly nonlinear circuit components is difficult due to the high instability, especially when selecting Pareto-optimal solutions (solutions along the Pareto front). The composition and synthesis of the analog circuits can take place at different levels of abstraction, and these are investigated in this project: i) the transistor level, ii) the elementary cell level, and iii) the network level.

Solving numerical and functional problems is well studied and understood with feedback (optimization error) from another analytical function. The feedback from circuit design either requires evolvable hardware [5] or an electronic simulator that computes the response of a circuit to an input stimulus for a given set of parameters (structure and component values), as used in this work. Simulation-driven multi-parameter optimization is increasingly applied at different levels of electronic design, not only to transistor sizing. An optimization-oriented framework for high-frequency electrical interconnect design, for example, has been presented in recent research, including optimization frameworks for the design of high-speed printed circuit boards by Xu et al. [6]. While this work operates at the passive component rather than transistor level, the methods discussed and evaluated in this work can be applied to broader optimization approaches, too, including structural and transistor-level optimization as a sub-part.

Gradient-based algorithms (gradient descent, GD) for solving parameter problems require derivatives of the target functions with respect to the parameters. Even a small-signal amplifier has at least four variable parameters. For analog circuits, as a function approximation, the gradient must be calculated numerically using electronic simulation (at least two simulations per parameter). Computational complexity is exploding rapidly. Stochastic algorithms, such as Genetic Algorithms (GAs) or Simulated Annealing (SA), on the other hand, require only one calculation per parameter set candidate, but sometimes show slow and non-reproducible convergence. Evolutionary and genetic algorithms can be used for the design and optimization of analog electronic circuits as an alternative to classical circuit design and as a solution to the optimization problem, especially for components with increased parameter scattering, cross-sensitivities, and drift (aging), since Monte Carlo simulation is an inherent component. Such circuits can also have increased fault tolerance, as Ji et al., using eevolvable hardware (EH), already showed [7], and Greenwood et al. [5] evaluated this for self-adaptive systems. GAs can be used for multivariate optimization problems with multiple target functions, as shown by Deb, and Azzouz et al. were able to show their suitability [8,9]. The problem of optimization in analog circuits lies in their high nonlinearity. When optimizing, it is necessary to strive for a Pareto optimum, i.e., that the improvement of one target function does not lead to the deterioration of another.

In principle, GAs can be understood as generative and parameterizing without a model and require only a small set of rules (boundary conditions) and simple crosslinkable elementary cells, which do not have to be ideal. GAs can thus be used for the optimization of structure and parameters. In this project, we consider both static structures with dynamic parameters (e.g., resistance values) and completely structurally and parametrically dynamic circuits (complete EH). With GA, the structure can be encoded in binary. The optimization and coding of numerical values (e.g., component values) are more difficult, but Bessaou et al. [10] propose a methodological adaptation of the GA method for numerical values.

Combinations of GAs and SA show improved convergence. Also, a combination of GD with SA, as described by Treadgoldn et al. with SARPROP [11] or SAGRAD by Bernal et al. [12], has been shown to be promising for solving unstable optimization problems, but due to the integrated electronic simulation, a high computational effort is required for the numerical calculation of the gradients. The stochastic algorithms, GAs/SA, have the advantage over the gradient-based ones, such as GD, that only one electronic simulation (or measurement on an EH platform) has to be performed per parameter set, whereas GD requires two n simulations for n parameters per parameter set in order to calculate the gradients numerically. In this work we compare GAs, SA, and GD used to optimize a small-signal amplifier. It turned out that these algorithms benefited from a model-supported approach in convergence and quality (here, boundary condition models for the components

derived from analytical models). The analog model was computably integrated into the optimization process by an integrated electronic simulator (ngspice) (in order to calculate the target functions). With these model-assisted algorithms, GA showed a consistently stable and fast convergence. GA has been modified for the optimization of numerical values (coding with atomic numerical values and algebraic recombination instead of bits and Boolean linking operations).

Another aspect is the robustness of the electronic circuits with respect to malfunction, aging, parameter variance in the components, and the failure of individual components. There are reports of improved stability and fault tolerance of GA-generated circuits compared to circuits from classical circuit design [13], which is important both technologically and with regard to the in-sensor approach. GA can be used at various levels of abstraction, such as at the transistor level [14,15], as described by Campilho-Gomes et al. for the synthesis of flat transistor circuits shown in [16], the sub-circuit level (OpAmp, neuron), and the mathematical level with symbolic regression [17].

Finally, GA and SA algorithms are suitable for parallelization with GGPU architectures, such as shown by Johar et al. [18] and Cheng et al. [19]. The challenge lies in the transfer of the evaluation function to GPU architectures, where the replacement model of an elementary cell from electronic simulation, e.g., as a KNN model, is assumed to be efficient and directly transferable. The ratio of the calculation time from the actual GA/SA operations to the calculation of the evaluation function can be estimated from 1:100 (with a KNN replacement model) to 1:100,000 (with electronic simulation).

Since the electronic simulation of analog circuits, an iterative solution problem in non-directional graphs with nonlinear components, is computationally intensive and the design times are significantly affected, purely numerical substitute models can be derived for certain model approximation architectures and basic circuits. An example is the calculation of the parameters of a KNN, where the product sum function can be replaced by a current sum amplifier and the sigmoid function by a modified function (see ref. [20]). The individual architectures considered in this project have different degrees of abstraction and possibilities of parametric adaptation. Replacement modeling is central and can be found at the electronic and circuit level, i.e., a physical and electronic simulation model of a transistor, as shown by Sideris et al. based on physical models, and Bornemann et al. based on experimental data [21,22], and on the other hand, reduced behavioral models of a sub-circuit (such as the use of a current-controlled 3-transistor OpAmp replacement model for analog KNN parameterization by Bosse et al. [20]), which can be integrated into the synthesis process. The challenges in determining a suitable circuit (structure and parameter set) are component variations, nonlinear transmission behavior, interaction of circuit parts, and environmental conditions (including temperature) that must be taken into account in the design process. By means of integrators as stateful models, time-dependent drift can be compensated; for example, a method which was described, inter alia, by Goyal et al. [23] has been investigated. The problem in temporal drift correction with technical integrators is their own drift behavior, which is further exacerbated with OECT electronics and must be compensated appropriately.

The goal of this work is the comparison of four algorithms (GD, SA, GAs, and PSO) to optimize the parameters of three circuit classes: 1. a small-signal amplifier with one transistor, 2. a simple operational amplifier with three transistors, and 3. a sigmoid function approximation circuit. Sure, these are trivial circuits, but the circuit classes 1 and 2 can be designed by solving electrical equations analytically, too. We have to solve a multivariate (resistor values) and multi-goal (static operating point, distortion, fixed gain, or a function error) optimization problem. We introduce modified GA/SA algorithms with added rule-based constraint models. This enhancement results in significantly better

convergence and training stability. We compare the performance of the four optimization methods for the electronic circuit with respect to the parameter variance in the different solutions and convergence. As a baseline experiment, we use a simple functional regression solved by the optimization algorithms, too. We evaluate and compare the component optimization for bipolar junctions (BJTs) and organic electro-chemical transistors (OECTs), differing significantly in transistor transconductance and transfer curves, which is expected to have an impact on convergence and parameter variance. All optimization methods are introduced with some basic background and the algorithms used to carry out the experiments. This is a simulation study, but uses data and models from real devices.

The structure of this work is as follows: All circuits and algorithms used in this work for circuit design are introduced and discussed in Section 2, showing the difference and particular requirements for circuit optimization. The core concept is the integration of the electronic simulator *ngspice* (*spice3f* derivate, originally published in 1973 by Nagel et al. [24]) into the optimization flow. The simulator requires component models, especially for the semiconductors, as discussed in Section 3. Section 4 describes all of the experiments carried out to investigate the behavior and quality of different optimization algorithms, which are finally evaluated in Sec. 5. We focus here on problems with fixed topology (because it is already known in advance), but the investigated methods can also be used for combined structural and component parameter optimization, and the results of this work are transferable.

The optimization methods discussed in this work are characterized by a varying number of hyper-parameters (e.g., update rates and random weight parameters). Exploring an exhaustive (nearly infinite) hyper-parameter space would lead to an explosion in the scope of this paper and this work. Therefore, we limited our experiments to suitable hyper-parameter sets for each method selected from the pre-experiments summarized in Section 4. There are two main evaluation metrics used throughout the work: 1. the average number of epochs (or more specifically, simulations) required to find a suitable solution satisfying the given constraints, and 2. the average termination rate of a method finding a suitable solution.

2. Methods

The basic workflow and concept are shown in Figure 1. A circuit is given here as a netlist of components (structure) and a parameter set defining the component values and environmental parameters, e.g., temperature. The parameter set consists of dynamic parameters (component values) and static parameters (e.g., voltages). The circuit has dedicated input and output ports (defining a stimulus–action model). This circuit is integrated into an optimization function or algorithm module. The goal of the optimizer is to find an optimal parameter set solution that satisfies multiple objective functions. The output of the optimizer, a parameter set solution, is fed into the circuit to evaluate the quality and fitness of a solution hypothesis.

In this work it is assumed that the circuit structure is given and static, i.e., the number of components and the connection of the components. The circuit is simulated using the ngspice electronic simulator [25] (version 44 with osdi model support) within a test bench, providing the supply and test voltage sources. The output of the electronic simulation (e.g., static DC analysis or frequency and distortion AC analysis) is processed by an analyzer that provides input features for the loss function computing a scalar loss (typically in the range [0, 1]) as a superposition from a set of objective functions *tf*. The loss output is used by the optimization algorithms to calculate and assess a solution with a new set of parameters (stochastic algorithms) or to compute a new set of parameters.

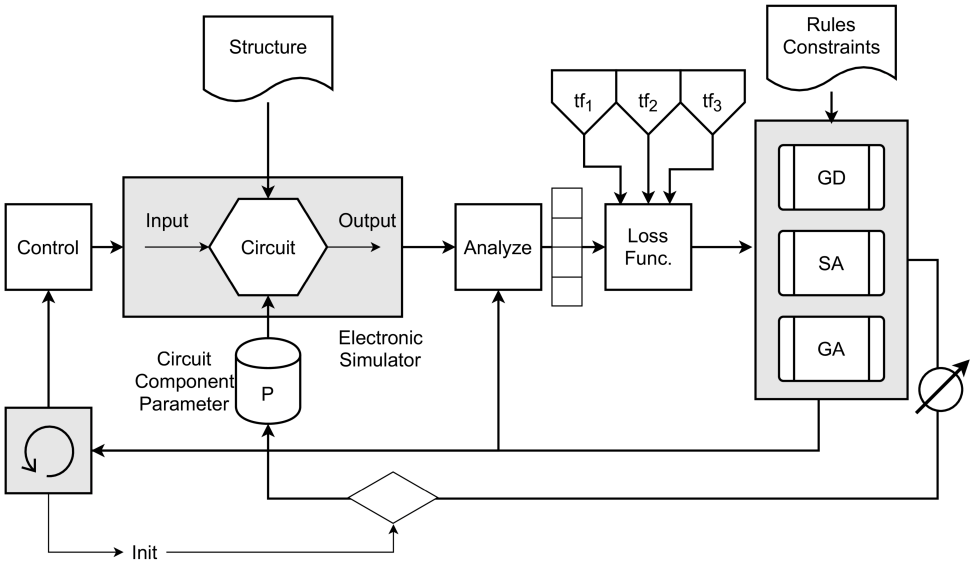

Figure 1. The basic concept consisting of an electronic simulator, a simulation analyzer, a simulation controller, and the optimization algorithms GD, SA, and GA (used mutually exclusively or in combination); *tf* are target functions defining the target attributes to be optimized.

2.1. Test Circuit 1: Small-Signal Amplifier

A typical test circuit is shown in Figure 2, for a bipolar transistor (BJT) and an organic electro-chemical transistor (OECT), as used in this work. It is a small-signal amplifier with one nonlinear component (BJT transistor) and four linear components (resistors), as well as two coupling capacitors. This simple amplifier is used as a first test circuit to evaluate and compare different parameter optimization algorithms. This amplifier circuit is characterized by a static DC operating point, which is commonly derived from transfer curves of the transistor by using analytical rules. Therefore, we can compare the parameter output of the optimization algorithms with the classical parameter calculation.

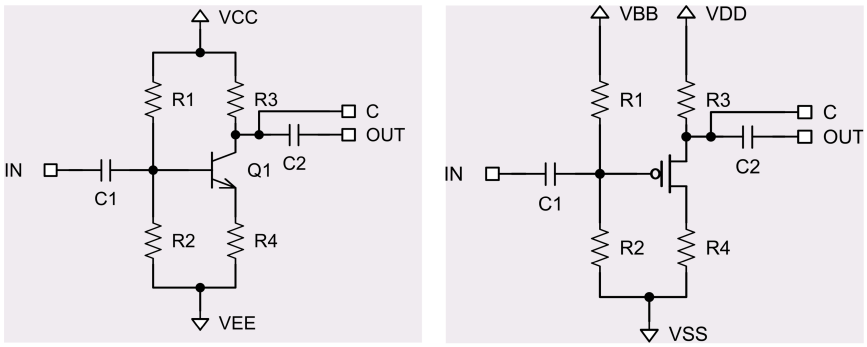

Figure 2. The small-signal amplifier circuit used in this work. (**a**) Q1 is a bipolar junction transistor; (**b**) Q1 is an OECT. C is a DC control point, and VBB is a separate bias voltage.

The small-signal AC behavior of the circuit is determined by the gain and the total harmonic distortion of an amplified sine signal:

$$G = \frac{V_{\text{out}}}{V_{\text{in}}}, THD = \frac{\sqrt{\sum_{n=1}^{\infty} V_n^2}}{V_0}, \quad (1)$$

with V_0 denoting the amplitude of the fundamental harmonic of the output voltage at frequency f_0, with V_1 denoting the amplitude of the second harmonic of the output voltage at frequency $f_1 = 2f_0$, and so on.

The initial parameter set is computed by a random process with random values within typical parameter ranges derived from analytical considerations, e.g., the typical value distribution from existing circuits or by analytical pre-computations.

To compare the parameter set computed by the following algorithms automatically, we derive a possible parameter set (i.e., the resistor values) of the amplifier analytically.

The classical way to set the static DC operating point of a transistor amplifier starts with the transistor curve diagram, as shown in Figure 3. Assuming a specific collector resistor value (and neglecting an emitter resistor), we choose the operating point approximately at the half supply voltage range, ensuring that the signal swing is within the active region of the transistor.

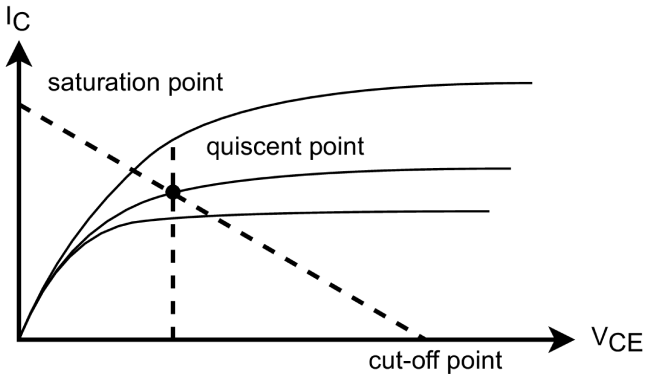

Figure 3. The operating point of a small-signal amplifier derived by the load line in the transistor curve diagram (collector current versus collector–emitter voltage).

The small-signal AC amplification within the operating range is given by:

$$G = -\frac{R_C}{R_E} \tag{2}$$

Without an emitter resistor R_E, the amplification is limited by the current amplification factor (β) of the transistor. The collector resistor R_C is chosen in advance. The value determines the output impedance of the amplifier, i.e., the impact of the load on the AC output voltage.

We can calculate the remaining resistor values at the operating point using:

$$\begin{aligned} &I_C = \tfrac{U_{RC}}{R_C} \approx \tfrac{U}{2R_C}, R_E \ll R_C \\ &R_E = \tfrac{R_C}{G} I_B = \tfrac{I_C}{\beta}, \\ &U_{B0} = U_{BE} + U_{E0}, U_{BE} \approx 0.7V, U_{E0} = R_E I_C \\ &R_1 = \tfrac{U - U_{B0}}{I_q + I_B}, I_q = m I_B \\ &R_2 = \tfrac{U_{B0}}{I_q}, \end{aligned} \tag{3}$$

with R_1 and R_2 as the bias network resistors connected to the base of the transistor, $R_C = R_3$ and $R_E = R_4$ in Figure 2, U_{BE} is the base-emitter voltage, I_B and I_C are the basis and collector currents, respectively, and m is a cross-current factor, here chosen as 5. With the chosen U = 10 V, β = 200, and R_C = 1000 Ω, we get R_E = 200 Ω for G = 5 (as set in our experiments), R_1 = 55 kΩ, and R_2 = 13 kΩ.

2.2. Test Circuit 2: Operational Amplifier

A second test circuit used for some selected experiments is a simple operational amplifier (OpAmp) consisting of three transistors and 10 resistors (3 fixed), as shown in Figure 4. The transistors J1 and J2 form a classical differential amplifier, with a current amplifier in the output stage.

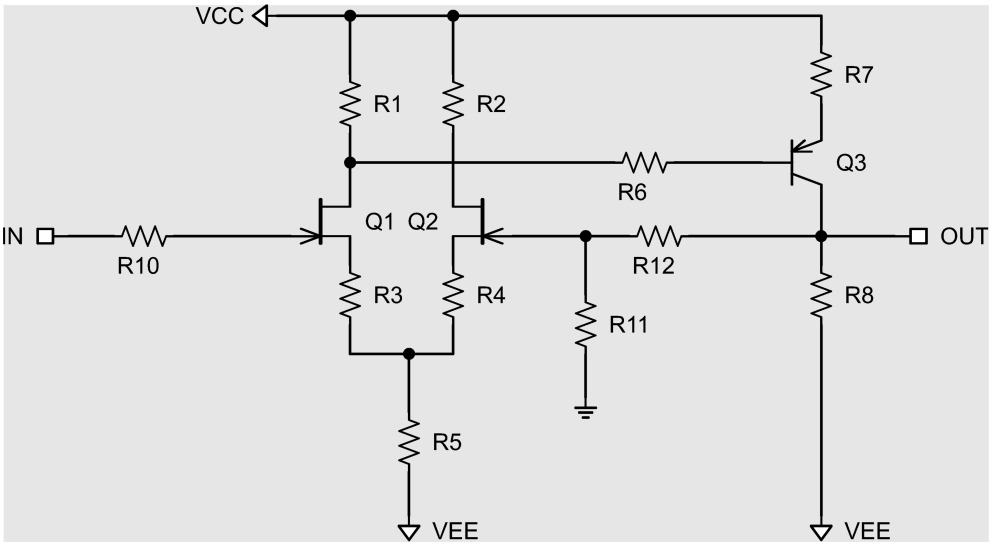

Figure 4. Minimal operational amplifier with three transistors, implementing a non-inverting amplifier. R10, R11, and R12 have fixed values.

The gain of the OpAmp is here given ideally by:

$$G = \frac{R_{12} + R_{11}}{R_{11}} \tag{4}$$

Due to the simplified and minimalistic design, there is deviation from the programmed gain, basically as a result of the low open-loop gain of the amplifier. The maximum and minimum output voltages are also limited (typically asymmetrically).

2.3. Test Circuit 3: Sigmoid Function

Previous circuits implement generic functions. The third test circuit should implement a specific mathematical function, here a sigmoid function, as shown in Figure 5, again consisting of three transistors, with a differential amplifier in the input stage and an emitter follower transistor circuit at the output. In contrast to the first two examples, this circuit must be optimized towards a mathematical function, i.e., defining the loss as the difference between the circuit input and the mathematical function:

$$f(x) = \frac{y_1}{1 + \exp(-w \cdot x)}, \tag{5}$$

with y_1 as the maximum value of the sigmoid function (normally 1, here a different value), and w as the slope of the sigmoid function. A reasonable slope range for the implementation using the circuit from Figure 5 is w = [0.5, 1]. High slopes require high amplifier gains, which cannot be achieved with the minimal implementation.

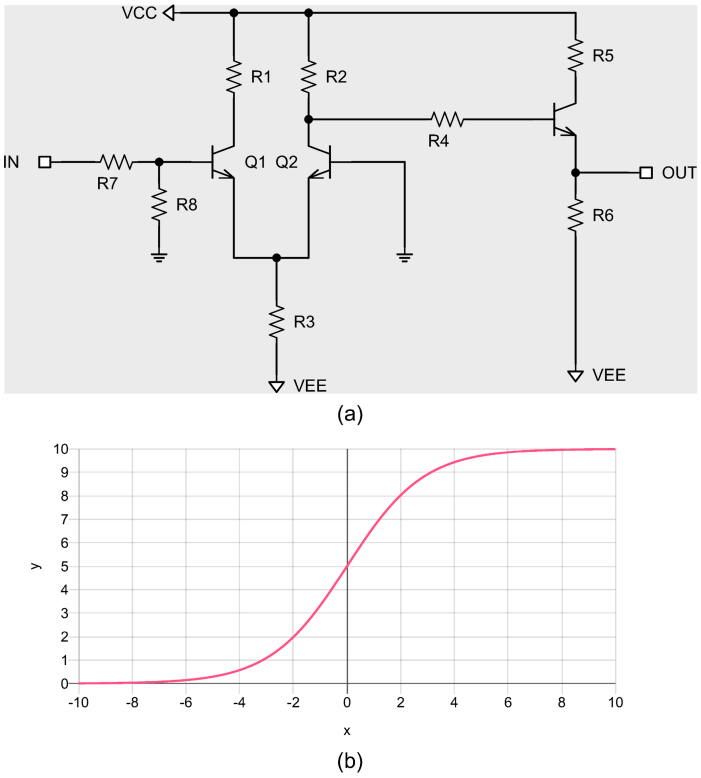

Figure 5. (**a**) The circuit that can implement the mathematical sigmoid (logistic regression) function and (**b**) the example plot of the sigmoid function with $y_1 = 10$, x = [−10, 10], and $w = 0.7$.

2.4. Gradient Descent Algorithm

The first optimization approach considered in this work is a classical error backpropagation algorithm using an error gradient descent (GD) approach, primarily used for data fitting. Gradient descent (GD) is a classical error-propagation algorithm which is deterministic in nature, in contrast to the metaheuristics used in this study [26]. In contrast to common numerical problems solved using GD on differentiable functions, e.g., those used in Machine Learning and Artificial Neural Networks, which are the sum of the product and the sigmoid functions for which an analytical gradient can be derived, in this work, the function to be optimized is a circuit, an undirected graph of linear and nonlinear components for which no closed-form gradient function can be derived. We require a numerical approach to compute the error gradient with respect to the circuit parameters, requiring an electronic simulator (e.g., SPICE) or configurable test hardware.

The electronic simulator is used for computing the loss and the numerical gradient of the loss with respect to the circuit parameters from the set P, as shown in Algorithm 1, which is rather simple and consists of one loop that terminates if the maximal number of epochs (n) or the loss threshold (lt) is reached. The update parameter α determines the progress and change in parameter values, normally assuming that the gradient has limited variations, which cannot be assumed with circuits.

The parameter initialization takes place with a parameter set P_0, which may be a zero vector, together with the permissible parameter ranges R, an optional data table $\boldsymbol{D}$ (used only for functional regression), the loss, model M, the gradient functions, and the optimization parameters. Two variants of GD exist. A data-driven version, *GD1,* evaluates the model against a labeled data table $\boldsymbol{D}$**,** performing a polynomial regression; a function-driven *GD2* obtains the loss from the simulator and is applied to the amplifier optimization of Figure 2, for which no labeled data exist. Unless stated otherwise, what follows concerns *GD2*.

Numerical gradient and cost. The derivative of the loss cannot be computed analytically; therefore, each component of the gradient is approximated by a central-difference formula [27]:

$$g_i = \frac{\partial l}{\partial p_i} \approx \frac{l(M, P + \delta e_i) - l(M, P - \delta e_i)}{2\delta}, \quad i = 1, \ldots, |P|, \tag{6}$$

with e_i being the i-th unit vector. The total number of simulations per iteration is two per parameter plus one for the loss, giving $2|P| + 1 = 9$ circuit model simulation runs for the four resistors. This is the major disadvantage of GD. In contrast, the stochastic algorithms need only one evaluation (single solution) or $|pop|$ evaluations (multiple solution population) per iteration. Even if GD needs only a low number of epochs, it has the most simulations per epoch if the parameter dimension increases, which makes it very expensive to run. Additionally, the accuracy of the gradient depends on the electronic simulation and its iterative solver, too, which can result in vanishing or exploding gradients. Choosing a suitable Δ_p is difficult.

The number of simulations can be reduced by using the asymmetric difference formula:

$$g_i = \frac{\partial l}{\partial p_i} \approx \frac{l(M, P) - l(M, P - \delta e_i)}{2\delta}, \quad i = 1, \ldots, |P|, \tag{7}$$

Here we need only $|P| + 1$ simulations for the gradients because all gradients share one term.

Update step. The candidate parameter is moved along the negative gradient and is clipped to keep it within the admissible range,

$$P \leftarrow \Pi_R(P - \alpha G), \tag{8}$$

where Π_R clips every component p_i to its range R_i. In GD1, the algorithm additionally carries the per-sample error $e = -\mathrm{EF}(M,P,D)$ as a factor, whereas in GD2, the raw numerical gradient is used.

Behavior on the circuit. GD struggles when the circuit gradient nearly vanishes (a plateaued region of the solution space) once the transistor leaves its active region. In such a situation, GD fails to converge, and whether a run makes progress at all also depends on whether its random initialization of the parameters places the operating point inside the narrow active window.

Smoothness of the loss. GD is the only algorithm used in this study that differentiates the loss, so it is the only one that constrains its form. A continuous loss function, as used in numerical problems, is suitable, but any conditional-tree-like loss functions (with multi-objective loss priorities) would fail: their discontinuities produce arbitrarily large derivative values unrelated to any descent direction. In contrast, the stochastic algorithms only sample the loss and are unaffected by this. The loss functions used for circuit optimization are discussed in detail in Section 2.8.

Variations of the standard GD with static update rates but dynamic gradient contribution are Adam, introduced by Kingma et al. [28], and AdaGrad, introduced by Duchi et al. [29], using dynamic (partially decaying) updates, providing commonly improved and faster convergence, and overcoming issues with low and high gradients. The Adam algorithm is shown in Algorithm 2, with three additional optimization parameters β1, β2, and ε. The epoch counter introduces a decay on the gradient contribution to the parameter adjustment. The Adaptive Gradient (AdaGrad) algorithm adds a gradient scaling by using the squared sum of all gradient values, as shown in Algorithm 3.

Algorithm 1: The core gradient descent algorithm. GD1 is used for a data-driven regression, and GD2 for electronic simulation (without labeled data). The input vector D is a training data table (for optimization of functional models)

```
function GD1(P₀,   // parameter vector, initial guess
          R,   // ranges or set of values
          D,   // data table (functional
          M,   // model passed to loss_function
          LF,  // loss function
          EF,  // error function
          GF,  // gradient function,
          lt,  // loss threshold,
          n,   // maximal number of iterations,
          α,   // update rate,
          δ    // delta for GF
          )
  P := P₀
  ∀ t∈{ 1,2,..,n} do
    ∀ u∈{ 1,2,..,|D|} do
      index = random(1,|D|)
      G = GF(M,P,D[index],δ)
      e = -EF(M,P,D)
      P := {q = pᵢ-α*e*Gᵢ , q ∈ Rᵢ, ∀ pᵢ ∈ P}
   done
   l = LF(M,P,D)
   if l < lt then break end
  done
end
function GD2(P₀,   // parameter vector, initial guess
          R,   // ranges or set of values
          M,   // model passed to loss_function
          LF,  // loss function
          GF,  // gradient function,
          lt,  // loss threshold,
          n,   // maximal number of iterations,
          α,   // update rate,
          δ   // delta for GF
          )
  P := P₀
  ∀ t∈{1,2,..,n} do
    G = GF(M,P,δ) // returns grad. vec. for each param., |G|=|P|
    P := {q = pᵢ-α*Gᵢ, q ∈ Rᵢ, ∀ pᵢ ∈ P}
    l = LF(M,P)
    if l < lt then break end
  done
end
```

In the experimental section, we will evaluate the GD algorithms for three different circuits with 4 and 8 parameters against stochastic algorithms, introduced in the next section.

Algorithm 2: Adam gradient algorithm

```
function ADAM(P₀,   // parameter vector, initial guess
          R,    // ranges or set of values
          M,    // model passed to loss_function
          LF,   // loss function
          GF,   // gradient function,
          lt,   // loss threshold,
          n,    // maximal number of iterations,
          α,    // update rate,
          δ,    // delta for GF
          β1,
          β2,
          ε
          )
  P := P₀
  M := {0, ∀ i ∈ 1..|P|}
  V := {0, ∀ i ∈ 1..|P|}
  ∀ t∈{1,2,..,n} do
    G = GF(M,P,δ) // returns grad. vec. for each param., |G|=|P|
    M = { β1*Mᵢ+(1-β1)*Gᵢ, ∀ i ∈ 1..|G|}
    V = { β2*Vᵢ+(1-β2)*Gᵢ², ∀ i ∈ 1..|G|}
    Mt = { Mᵢ/(1-β1ᵗ), ∀ i ∈ 1..|G|}
    Vt = {Vᵢ/(1-β2ᵗ), ∀ i ∈ 1..|G|}
    P := {q = pᵢ-α*Mtᵢ/(sqrt(Vtᵢ)+ε), q ∈ Rᵢ, ∀ i ∈ 1..|P|}
    l = LF(M,P)
    if l < lt then break end
  done
end
```

Algorithm 3: Adaptive AdaGrad algorithm

```
function ADAGRAD(P₀,   // parameter vector, initial guess
          R,    // ranges or set of values
          M,    // model passed to loss_function
          LF,   // loss function
          GF,   // gradient function,
          lt,   // loss threshold,
          n,    // maximal number of iterations,
          α,    // update rate,
          δ     // delta for GF
           )
  P := P₀
  ∀ t∈{1,2,..,n} do
    G = GF(M,P,δ) // returns grad. vec. for each param., |G|=|P|
    s = ΣG²
    P := {q = pᵢ-α*Gᵢ/sqrt(s), q ∈ Rᵢ, ∀ pᵢ ∈ P}
    l = LF(M,P)
    if l < lt then break end
  done
end
```

2.5. Simulated Annealing Algorithm

In contrast to the deterministic pure functional gradient error propagation algorithm, Simulated Annealing (SA) is a pure stochastic process-driven approach. It is a suitable optimization method when global optimization is required, the space of the objective function is complicated and noisy with many local minima, and when the objective function has a lot of parameters and/or is non-differentiable, which is the case in circuit optimization. The main advantage of a pure stochastic process is that no gradient has to be calculated, reducing the number of model evaluations to one per iteration, compared with the GD algorithm, and reducing the risk of vanishing gradients. The disadvantage can be uncontrolled parameter space exploration with poor convergence stability. The model itself does not contribute to parameter modification.

Simulated Annealing (SA) is often referred to as one of the earliest metaheuristics and one of the initial algorithms to have a clear plan to avoid running blind into a local minimum/maximum [30]. The inspiration for this algorithm comes from the work of Metropolis et al. in statistical mechanics [30].

It is a type of single-state metaheuristic method, as the algorithm chooses one random candidate solution out of the solution space as a starting point. New candidate solutions are generated by slightly shifting the current candidate solution. If they reduce the loss, then they are accepted; otherwise, they are either rejected or accepted based on a separate acceptance mechanism. This acceptance mechanism, guided by a control temperature T with a higher value during early iterations of the algorithm, is designed to accept worse candidate solutions with greater probability. As the iterations progress, the temperature T is reduced, and the probability of accepting worse solutions decreases. The temperature term is reminiscent of the physical process of cooling of solids, where the particles within move to a low-energy ground state (optimum solution).

There are two major advantages of using SA over GD. First, no derivative of the loss with respect to P needs to be calculated, which ensures that only one evaluation of the loss function takes place in each iteration. The GD-based optimizer needs a total of 5 (9) simulation evaluations per iteration for the 4 resistor values. Comparing both algorithms based on per-iteration cost alone means that SA is around 8 times cheaper than GD. Second, the solution space contains plateaued regions where GD would simply fail to advance irrespective of the update rate α (Figure 6). At any local minimum, GD would still manage to converge and terminate. In contrast, SA remains unaffected in such a situation, as it is derivative-free and its acceptance mechanism is designed to counter this.

Generation of candidate solutions. A candidate is formed by perturbing (adding noise to) all $|P|$ components of the current parameter vector simultaneously. A candidate is generated as

$$P_{\text{new}} = \{q = p_i + \text{NF}(\theta),\ q \in R_i,\ \forall p_i \in P\}, \tag{9}$$

where NF is the random number generator, $\theta \in [0, 1]$ is the random contribution that controls the magnitude of the perturbation, and R_i is the admissible range given for parameter i. Every component q of the new candidate is constrained to its range, so that all candidates remain inside the search space. Since SA keeps a population of one, a single iteration requires exactly one proposal and one evaluation of the loss function. The stochastic modification of parameters commonly assumes that the range of the parameters is similar, which is not the case in analog circuits.

The SA algorithm used in this work is shown in Algorithm 4. We enhance the standard SA algorithm with back-tracking (falling back to an older but better solution, shown in lines 36–40, and relying on snapshots generated in lines 23–28), limiting the range of parameters by searching new parameter values as long as the parameter is outside of its range. The idea is the creation of a new solution by adding small noise to the parameter values via a random process. The new solution is evaluated, and if the loss is lower than that of

the previous solution, the new parameter set is adopted as the current best solution. If the new loss is not better than the old, a random process decides whether to adopt the solution. Because no gradient must be evaluated, different objective and loss functions can be considered individually or prioritized (using a decision tree).

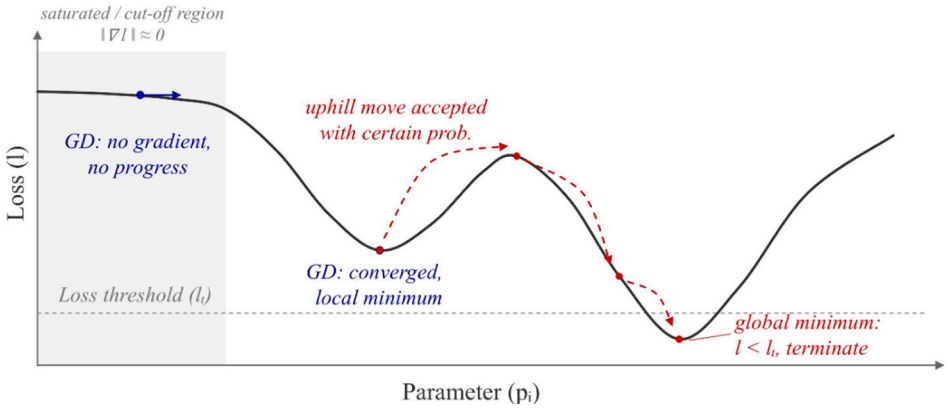

Figure 6. The schematic one-dimensional parameter space, showing the advantage of SA over GD and the plateaued region where GD shows no progress. GD gets trapped in a local minimum, whereas SA manages to escape by accepting an uphill move, with a probability given by the acceptance criteria.

Acceptance criteria. The loss value for the current best solution is denoted by l_{best}, and l_{new} denotes the loss value for the candidate. The candidate is accepted unconditionally if $l_{\text{new}} < l_{\text{best}}$; otherwise, it is accepted with a probability given by one of the following two methods:

$$p_1 = \begin{cases} 1 & P(0,1)T > r \\ 0 & \text{else} \end{cases}, \quad r = \frac{l_{\text{best}}}{l_{\text{new}}}$$
$$p_2 = \begin{cases} 1 & P(0,1) < \exp\left(-\frac{|l_{\text{best}} - l_{\text{new}}|}{T}\right) \\ 0 & \text{else} \end{cases} \tag{10}$$

Here, p_1 is the normalized method, and p_2 is the non-normalized method. The non-normalized method is initialized with a temperature T_0 scaled to the loss ranges, whereas the normalized method uses $T_0 = 1$ and is independent of the possible loss ranges but sensitive to the loss-progress gradient. The non-normalized method is called the Metropolis criterion [31].

The purpose of accepting a worse candidate is to let the search escape a local minimum. A rule that accepted only improvements would certainly stop at the first minimum it entered, but an uphill move offers an escape route. Both expressions are evaluated only in the branch $l_{\text{new}} \geq l_{\text{best}}$, since a candidate with $l_{\text{new}} < l_{\text{best}}$ is already accepted unconditionally. The temperature is used to control the permissiveness of the rule. The search behaves like a random walk while T is high, and approaches a greedy descent as T trends to zero.

Cooling schedule. The temperature is reduced geometrically at every iteration using the following formula:

$$T_k = T_0 \lambda^k, \quad \lambda \in (0,1), \quad k = 1, 2, \ldots, n, \tag{11}$$

with T_0 being the initial temperature value, λ the cooling rate, k ranging from 1 to n, and n the number of iterations. The temperature is multiplied by a constant factor λ at every iteration and reaches $T_n = T_0 \lambda^n$ at the end of the permissible iterations. The cooling rate guides the search convergence within the available number of iterations. A value close to 1 keeps the temperature high throughout, which means that the worse candidates continue to be accepted

and the search stays an unguided random walk. A small value freezes the temperature after just a few iterations, which means that the remaining iterations are effectively a greedy descent.

Algorithm 4: Core SA algorithm with optional back-tracking

```
1: function SA(P₀,   // initial parameters
2:             R,  // ranges (discrete set) of vector
3:             M,  // model passed to loss_function
4:             LF, // loss function
5:             lt, // loss threshold for termination
6:             n,  // maximal number of iterations
7:             θ,  // random contribution [0-1]
8:             NF, // random number generator
9:             λ,  // cooling rate
10:            T₀, // initial temperature
11:            am,   // two methods to accept new solution
12:            bt)   // enable back tracking?
13:  P := P_best := P₀; T := T₀; failed := 0;
14:  l_best := LF(M,P)
15:  ∀ i∈{1,2,..,n} do
16:    decr T by λ
17:    P_new = {q = p_i + NF(θ), q ∈ R[i], ∀ p_i ∈ P}
18:    l_new = LF(M,P_new)
19:    if l_new < l_best or
20:        (am = 1 and runif()*T > l_best/l_new) or
21:        (am = 2 and runif() < exp(-|l_best-l_new|/T)) then
22:      P := P_new
23:      if l_new < l_best then
24:        P_best := P_new
25:        l_best := l_new
26:        failed := 0
27:        incr stats.accept₁
28:      else
29:        incr failed
30:        incr stats.accept₀
31:      end
32:    else
33:      incr failed
34:      incr stats.reject
35:    end
36:    if bt ≠ 0 and failed ≥ bt then
37:      P    := P_best
38:      failed := 0
39:      incr stats.back
40:    end
41:    if l_best < lt then break;
42:  done
43:  return (P_best,l_best)
44: end
```

Back-tracking. A counter of consecutive, non-improving iterations is maintained. When it reaches a threshold b_t, the current candidate is reset to the incumbent best solution, and the counter is cleared. This is not part of the classical SA algorithm. It intensifies the search around the best solution found so far, assuming that a long run of consecutive rejections means that the current candidate is trapped in an inconsequential region of the parameter space. It restores the position but not the temperature, which means that the exploration capability already removed by the cooling schedule is not recovered. The mechanism is optional and can be disabled by setting $b_t = 0$.

Table 1 lists the settings used for the experiments of Section 4. Each configuration was repeated $\langle N \rangle$ times with independent random initializations; all averages, variances, and termination rates reported in Section 5 are taken over those repetitions.

Table 1. Parameterization of the SA algorithm.

Symbol	Meaning	Default Value
T0	Initial temperature	1 [a.u.]
λ	Cooling rate	0.999 [a.u.]
θ	Proposal amplitude	0.05 [a.u.]
am	Acceptance rule	1/2 [a.u.]
bt	Back-tracking threshold	0/100 [a.u.]
n	Maximum iterations	1000
lt	Loss threshold	Problem specific
R1 . . . R4	Admissible ranges	Problem specific
N	Runs per configuration	1

2.6. Genetic Algorithm

The SA algorithm described in Section 2.5 works with a single candidate solution in each iteration, which limits its ability to explore the complete solution space and, thus, is prone to settling in a local minimum. A genetic algorithm, on the other hand, explores the solution space with multiple paths by using a population of candidate solutions in each iteration. This large and diverse population keeps improving generation by generation, which helps the algorithm in extensive exploration and avoiding local minimum points, enabling a higher parameter space variation in the creation of new solutions and selection process. It borrows the idea from biological evolution [32]: the fitter individuals from a diverse population are more likely to produce fitter offspring. In each generation (iteration), the algorithm evaluates the population, selects parents, recombines them (crossover), mutates the offspring, and replaces the old population (see Figure 7). This is repeated until the loss is small enough or the maximum number of iterations is reached. Classical genetic algorithms encode a solution as a binary string, but here we use a real-valued encoding [29]. Mutation adds random noise to the candidate values instead of flipping bits, and crossover mixes values instead of swapping bit substrings.

The GA is the most complex algorithm considered in this work, requiring a set of helper functions, as shown in Algorithm 5, and is finally used by the main GA shown in Algorithm 6. There are a lot of variations of GAs, and a GA is composed of these helper functions, e.g., different randomized selection functions and the crossover function. The specific function selection is part of the hyper-parameter space of the optimization problem and can depend on specific problems. The corresponding principal generational cycle of the genetic algorithm is shown in Figure 7, driven by a set of parameters (basically, the population size and stochastic parameters, as shown in Algorithm 6).

GA optimization was initially used for binary or discrete variables for solving structural and combinatorial problems, e.g., by finding connections among components. Here we have the optimization of real-valued variables, constrained by given ranges. Therefore,

the mutation of a solution uses random noise instead of bit changes, and the crossover function performs a random scaled linear combination of parameter variables under range constraints, similar to differential evolution.

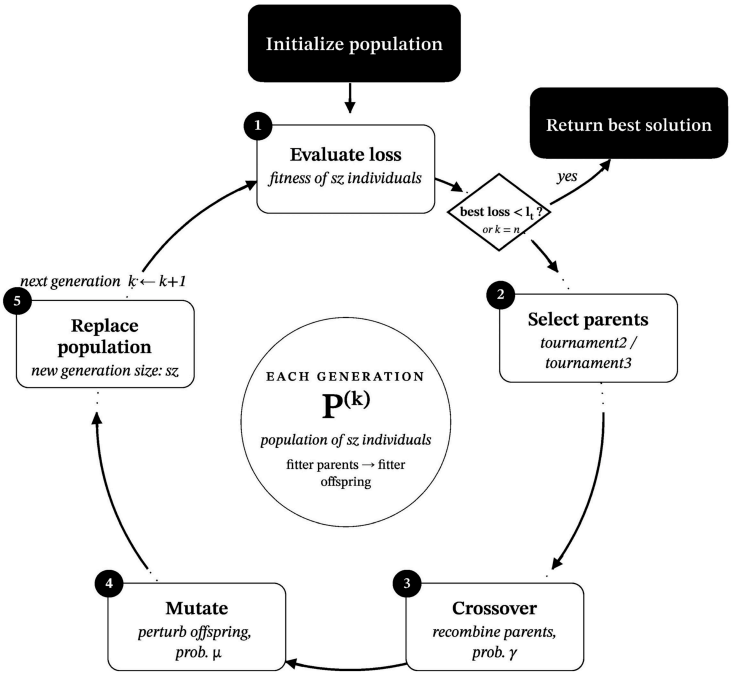

Figure 7. The generational cycle of the genetic algorithm. An initial population of sz candidate solutions is seeded within the parameter ranges R, then repeatedly refined: each generation is evaluated, parents are chosen by tournament selection, recombined by crossover (prob. γ), and perturbed by mutation (prob. μ), and the resulting offspring replace the old population. The loop halts once the best loss falls below l_t or after n generations.

The initial population is drawn randomly within the ranges R. R_i is the distance between the lower and upper bound of range i. The population size is denoted by sz,

$$P_j^{(0)} = (p_1, \ldots, p_{|P|}), \quad p_i \sim U(\min R_i, \max R_i), \quad j = 1, \ldots, sz. \tag{12}$$

Selection is done via tournament (tournament2 or tournament3) [33]: two or three individuals are picked at random, and the fittest (with the lowest loss) of them is kept as a parent. Selection does not depend solely on the loss values of the whole population but only on those of the candidates selected for the tournament. This makes sure that even suboptimal candidates might get a chance to generate new candidates by mixing with top candidates, which pushes the optimizer out of local minimum traps. For crossover (with probability γ), two parents a and b are taken, one coordinate i is picked at random, and a weight $r \sim U(0, 1)$ is drawn. New candidates are simply interpolated values between the parents so they stay within the range as well.

$$\begin{aligned} c &= a, \quad c_i = b_i + (a_i - b_i)r, \\ d &= b, \quad d_i = a_i + (b_i - a_i)r. \end{aligned} \tag{13}$$

Mutation (with probability μ) perturbs a candidate solution by adding uniform noise scaled to the width R_i of its range and keeps the result inside the range.

$$\begin{aligned} &\text{mutate}(P)_i = \text{clip}(p_i + u_i, R_i), \\ &u_i \sim U\left(-\tfrac{1}{2}\|R_i\|, +\tfrac{1}{2}\|R_i\|\right) \end{aligned} \tag{14}$$

Scaling by $|R_i|$ provides each parameter a step proportional to its own span. This is important as the model parameters, i.e., resistances, vary a lot in terms of values.

GAs have significant running costs as compared to SA since each iteration costs sz simulations at about 100 ms each, against one per iteration for SA. The sz evaluations of an iteration are independent of each other and can run in parallel. The hyper-parameters and typical values are shown in Table 2.

Table 2. The typical parameterization of the GA.

Symbol	Meaning	Value
sz	Population size	10
γ	Crossover probability	0.3
μ	Mutation probability	0.3
–	Selection operator	⟨tournament2/tournament3/fittestRandom⟩
n	Maximum generations	250
m	Results returned	10
lt	Loss threshold	0.1
R	Admissible ranges	Given by circuit

Each iteration requires N model computations for the loss computation, i.e., in the case of electronic circuits, a simulation run with a population size of N.

2.7. Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a population-based method, similar to the genetic algorithm. It works with a population (a group) of candidate solutions and keeps improving them in each iteration. The difference lies in the way a new solution is produced. In the case of GA, a new solution is generated by mutation, crossover, and selection, whereas in PSO, each candidate is a particle that moves through the parameter space, getting pulled towards the best solution it has found itself and towards the best solution found by the other particles that inform it [34]. Another difference compared to previous algorithms lies in range constraints. The other algorithms described before were extended by explicit parameter range constraints, whereas PSO carries these constraints inherently. The progress of the previous stochastic algorithms were sensitive to parameter ranges, which is expected not to be the case with PSO assessing the relative progress (parameter gradient) of each solution.

Particle state. In the swarm, every particle i holds a position pos_i and a velocity vel_i, and pos_i is one complete parameter set (a point in the search space), whereas a velocity vel_i is the step the particle will take next. Every particle also remembers its own best position so far, given by $posb_i$. Other particles communicate the best position found among them so far, which is given by $posnb_i$. The single best position found by the whole swarm is saved as $gbest$. The loss is evaluated once per particle and based on these values, the best records are updated.

Velocity and position update. In each iteration, the velocity of each particle is updated, after which it moves. The new velocity keeps a fraction (calculated by inertia weight w) of the old velocity. Moreover, two pulls are added to it. One is directed towards the particle's own best position (weighted by c_1), and the other is directed towards its neighborhood best (weighted by c_2).

Algorithm 5: Helper functions for GA

```
// random initialization of vector within ranges
function seed (n,R)
  return {x_i ∈ R_i, i=1,2,..,n}
end
// random modification of vector within ranges
function mutate (p,R)
  return {x_i = p_i+runif(-0.5,0.5)*||R_i||,x_i ∈ R_i, i=1,2,..,|p|}
end
// linear element-wise cross-over of vectors
function crossover (a,b,R)
  c = copy(a)
  d = copy(b)
  i = runif(1,|a|)
  r = runif(0,1)
  c[i] := b[i]+(a[i]-b[i])*r
  d[i] := a[i]+(b[i]-a[i])*r
  return (c,d)
end
// Select 1
function tournament2(pop)
  n = |pop|
  a = pop[runif[1,n]]
  b = pop[runif[1,n]]
  return bestfit(a,b)
end
function tournament3(pop)
  n = |pop|
  a = pop[runif[1,n]]
  b = pop[runif[1,n]]
  c = pop[runif[1,n]]
  return bestfit(a,b,c)
end
function random(pop)
  n = |pop|
  return pop[runif[1,n]]
end
function fittest(pop)
  // population is sorted by fitness, take first element
  return pop[1]
end
// Select 2
function select2(select1,pop)
  return {select1(pop),select(pop)}
end
```

Algorithm 6: Core GA

```
function GA(
      sz, // popoulation size
      λ, // crossover rate
      μ, // mutation rate
      n, // iterations
      m, // maximum number of results
      LF, // loss-fitness function
      lt, // loss threshold for termination
      SL1, // Select one entity from population
      SL2, // Select two entities from population
      CF,  // Crossover function
      MF,  // Mutation function
)
P={seed(i),i=1,2,..,sz} // initial population
∀ i∈{1,2,..,n} do
   pf = sort {(fitness=LF(p),entity=p), ∀ p ∈ P}
        ascending by fitness score
   pf_mean = mean({p.fitness, ∀ p ∈ pf})
   pf_dev = sd({p.fitness, ∀ p ∈ pf})
   r = pf[1].fitness < lt
   if r then return pf
   else
    P_new = {} // new population
    if fittestAlwaysSurvive then P_new:=P_new+{pf[1]} end
    if runif(0,1) < λ and |P_new| < sz-1 then
       children = CF(SL2(pf)
       if runif(0,1) < μ then
          children:={MF(e), ∀ e ∈ children} end
       P_new:=P_new+children
    else
     child = SL1(pf)
     if runif(0,1) < μ then child:=MF(child) end
     P_new:=P_new+{child}
    end
    P := P_new
  end
end
pf = sort {(fitness=LF(p),entity=p), ∀ p ∈ P}
     ascending by fitness score
return pf
end
```

$$\begin{aligned} vel[i] &\leftarrow wvel[i] + c_1 r_1 (posb[i] - pos[i]) + c_2 r_2 (posnb[i] - pos[i]), \\ pos[i] &\leftarrow pos[i] + vel[i], \end{aligned} \tag{15}$$

with r_1,$r_2 \sim U(0, 1)$. The constant c_1 represents the strength of attraction a particle experiences for its own best position, and c_2 represents how strongly it follows its neighbors. After each particle's movement, every component of pos_{i} is kept within the bounds [r_{low}, r_{high}], thus making the range constraint inherent to the method. The velocity update is

shown in Figure 8. The contour lines in the example plot represent a fitness landscape across two parameters θ_1 and θ_2.

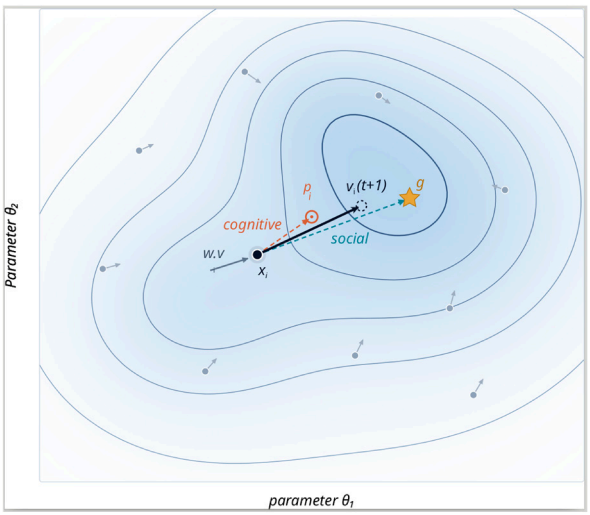

Figure 8. A schematic representation of the particle swarm optimization algorithm. Each particle i (black dot) traverses the parameter landscape by summing three influences: its retained momentum (inertia), an attraction toward its own best-known position p_i ("cognitive"), and an attraction toward the swarm's best-known position g ("social"). The contour lines and shading indicate the cost surface; the swarm collectively descends toward the global optimum.

To summarize, the PSO problem is defined by the following parameters, as used in Algorithm 7:

- The parameter vector dimension *dim*;
- Lower and upper bound ranges of each parameter, r_{low} and r_{high}, respectively;
- The particle population size *size*;
- A loss function *loss*;
- Two constants c_1 and c_2 controlling the optimization;
- Two weight parameters w_{min} and w_{max};
- A neighborhood size n_{ng};
- A neighborhood and weight optimization strategy.

The principle PSO algorithm variant used in this work is shown in Algorithm 7. PSO is basically intended for continuous optimization problems, although there are attempts to apply PSO to discrete problems, too.

Inertia weight. The value of the inertia weight w is not constant. It starts with a high value (near w_{max}), so that during the earlier runs, the particles get a chance to explore the parameter space. It is later reduced linearly towards the w_{min} value so that the swarm gets a chance to settle and refine (exploitation of the parameter space) [35]. The function *InertiaLinDec* (Algorithm 7) applies this decrease over the first three quarters of the iteration budget. It then holds w at w_{min} for the remaining iterations. This is similar to the explore-first and refine-later idea of the cooling schedule provided in SA, which is expressed here through weight rather than temperature.

Neighborhood. The communication concerning which particles share their best position is decided by a matrix denoted by *comm*. An entry $comm_{\text{i,j}} = 1$ means that particle i informs particle j. The function *Inform* is responsible for reading this matrix and copying the best position among the particle's informers into *posnb*. Three methods of communication are available [36].

The ring topology (*CommInitRing*, Algorithm 7) connects each particle only to its left and right neighbors. This makes sure that the good solutions spread slowly and the swarm is less likely to collapse to a single point too early. The random topology (*CommInitRandom*, Appendix A) allows each particle to inform randomly chosen particles. The links are redrawn whenever an iteration brings no improvement (*InformRandom*). The global topology (*InformGlobal*, Appendix A) connects all particles, allowing every particle to be pulled towards the single swarm best. This spreads the information the fastest but explores the least.

Cost and comparison. Since the loss is evaluated once per particle, one iteration means that the number of loss evaluations is equal to the size of the swarm. This is similar to the genetic algorithm, whose iteration costs one simulation per population member. In contrast, there is a single evaluation per iteration of SA and several evaluations of GD.

Algorithm 7: The pseudo code of the PSO algorithm (solve) with helper functions. Ψ is the settings table (static optimization parameter set). The variations in the communication and inform functions can be found in Appendix A

```
// settings
Ψ = {
    c1,c2: number, // constants
    wmin,wmax:number, // weight range
    dim : number, // parameter vector size
    nng : number, // neighborhood size
    size : number, // particle swarm size
    rlow,rhigh: number Array[dim], // ranges of parameter
}

// calculate linearly decreasing inertia weight (default)
function InertiaLinDec (step, Ψ)
   dec_stage = 3 * int(Ψ.steps / 4);
   if step ≤ dec_stage then
     return Ψ.w_min + (Ψ.w_max - Ψ.w_min) *
          (dec_stage - step) / dec_stage;
   else
     return Ψ.w_min;
   end
end
// general inform function based on the connectivity matrix comm,
// it copies the best position of the informers of each particle
function Inform (comm, posnb,posb,fitb,Ψ) // inform
  ∀j ∈ {1,2,..,Ψ.size} do
    bn = j // self is best
    ∀i ∈ {1,2,..,Ψ.size} do
      if comm [i][j] = 1 and fitb[i] < fitb[bn] then
        bn := i
    done
```

Algorithm 7: *Cont.*

```
30:    pos_nb[j]=clone(pos_b[b_n])
31:  done
32: end
33: // ring topology: choose informers
34: function CommInitRing (comm, Ψ)
35:  // comm is a communication ring matrix connecting left&right
36:  comm = 0
37:  ∀i ∈ {1,2,..,Ψ.size} do
38:    comm [i,i] := 1 // diagonal
39:    // particle i informs left and right
40:    comm [i,i-1] := 1, comm [i,i+1] := 1, if i>1 ∧ i<Ψ.size
41:    comm [i,Ψ.size] := 1,           if i=1
42:    comm [i,1] := 1,       if i=Ψ.size
43:  done
44: end
45:
46: function solve(loss, Ψ)
47:   error = MAX, gbest = Array(Ψ.dim)
48:   // information matrix
49:   // rows: information source
50:   // cols: information destination
51:   comm = Array(Ψ.size,Ψ.size)
52:   // position, velocity, best position,
         best informed position matrix,
53:   pos,vel,pos_b,pos_nb = Array(Ψ.size,Ψ.dim)
54:   fit, fit_b = Array(Ψ.size)
55:   improved = 0 // whether solution->error was improved
56:
57:   CommInitRing(comm, Ψ) // or CommInitRandom, see Appendix
58:
59:   // 1. Swarm initialization
60:   ∀i ∈ {1,2,..,Ψ.size} do
61:   ∀j ∈ {1,2,..,Ψ.dim} do
62:     a,b = runif(Ψ.r_low[j],Ψ.r_high[j])
63:     pos[i,j] := a
64:     pos_b[i,j] := a
65:     vel[i,j] := (a-b)/2
66:   done
67:   fit_b[i] = fit[i] = loss(pos[i])
68:   if fit[i] < error then
69:     error := fit[i]
70:     gbest[i] := pos[i,*]
71:   end
72:  done
73:  // 2. Solver
74:  return (error,gbest)
75: end
```

2.8. Loss Function

In electronic circuit design, we have multiple target functions that must be optimized, optimally finding a Pareto-optimal parameter solution. In our experiment using the single transistor amplifier, we want to find an optimal component value set with respect to three or four coupled loss functions (or expressions) used for the circuit optimization based on measured variables (from simulation):

1. The DC operating point (measured at tap C of the circuit), *VC*, commonly half of the supply voltage;
2. The AC output voltage with a gain *G*, reaching a given gain G_0;
3. The total harmonic distortion factor THD;
4. The output impedance of the amplifier (optional).

The different target and loss functions are not independent and nonlinearly coupled. The solution space could be narrowed with a number of separated solution islands.

We use a static point DC analysis and a transient AC analysis with harmonic distortion analysis.

$$\begin{aligned} l_1 &= \left(1 - e^{\min(0, V_C - V_{C,\min})}\right) + \left(1 - e^{\min(0, V_{C,\max} - V_C)}\right) \\ l_2 &= THD \\ l_3 &= \frac{G_0 - G}{G_0} \\ l_4 &= \frac{Z_0 - Z}{Z_0} \end{aligned} \tag{16}$$

The final loss calculated from all three objective functions must be accumulated and normalized carefully. The l_1 loss has an upper bound of 1 and is already normalized, whereas l_2 has no upper bound, and l_3 has an upper bound of G_0.

A typical loss accumulation used by all optimization algorithms is a sum of the losses:

$$l = l_1 + \tanh(l_2) + l_3 + l_4 \tag{17}$$

We normalize the THD (as a loss value) with the *tanh* function, which ensures a bounded value range $[-1, 1]$.

Alternatively, a conditional tree expression can be used to switch between the loss expressions depending on their relevance, but it introduces discontinuities, which can cause GD to fail, for example:

$$l = \begin{cases} l_1 & \text{if} \quad l_1 > 0.1 \quad \text{else} \\ l_3 & \text{if} \quad l_3 > 0.1 \quad \text{else} \\ & \text{else} \quad l_2 \end{cases} \tag{18}$$

The output impedance (generator resistance) of the circuit is measured with an additional load resistance connected behind the isolating capacitor (R_L = 1000 Ω):

$$\begin{aligned} Z_{out} &= \frac{R_L}{t} - R_L \\ t &= \frac{V_{out,L}}{V_{out}} \end{aligned} \tag{19}$$

The output impedance is roughly given by the sum of the collector and emitter resistors ($R_C = R_3 + R_E = R_4$).

The continuous loss functions are aggregates of losses for different objectives, whereas the tree-like loss functions can be used to add priorities to the optimization process, as deployed by some others (e.g., Campilho-Gomes et al. [16]). For the sake of simplicity, we will evaluate all optimization algorithms only by using aggregate loss functions.

Basically, the same loss functions are applied for the optimization of the discrete OpAmp (circuit 2), except for the target DC output voltage (at zero difference input voltage),

which should be set to 0 V. The optimization of the functional approximation circuit 3 uses a classical mean squared error from data points along the input range of the function, i.e., the sigmoid function, explained in detail in Section 4.

3. Models

The optimization of circuits investigated in this work consists of linear (ideally, resistor and capacitor) and nonlinear (transistor) components. The highly nonlinear transistor models and their transfer functions can have a significant impact on the optimization progress and, therefore, should be considered in detail here.

The transconductance defines the intrinsic transistor gain, and it is expected that the transconductance has a significant impact on the solving of optimization problems. The transconductance of a transistor is defined for current-controlled current sources (CCCSs), as well as for voltage-controlled current sources (VCCSs), by using:

$$\begin{aligned} g_m^{\mathrm{CCCS}} &= \frac{\Delta I_{\mathrm{out}}}{\Delta I_{\mathrm{in}}} \\ g_m^{\mathrm{VCCS}} &= \frac{\Delta I_{\mathrm{out}}}{\Delta V_{\mathrm{in}}} \end{aligned} \tag{20}$$

But it is important to note that the transconductance is the maximum slope of the transistor curve at a specific operating point, lying often in the highly nonlinear area, not suitable for a linear amplifier. The linear region of the transistor curve can have a much lower slope and transconductance.

For electronic simulation, a numerical model is required. For current-controlled devices, it is an $I_1(I_2)$ like function; for voltage-controlled devices, it is an $I_1(V_2)$ function. This transfer function is typically split into quadrants depending on input and output variables, the device type, and physical laws, defining a primary operating range, basically given by an incomplete function derived from physical laws and experimental data. Outside the operating range, transistor devices are normally not considered (except, e.g., describing break-through effects). For electronic simulation, we need a complete transfer function defined in all quadrants.

3.1. BJT

The primary transistor model we deploy in our experiments is the bipolar junction transistor (BJT) device model. We use the parameterized spice built-in BJT model. The BJT model is basically a current-controlled current source; these functions can be summarized according to Maadi [37] and split into base, collector, and emitter currents, which are entangled:

$$\begin{aligned} I_B &= I_{BE} + I_{BS} \\ I_C &= I_{CE} - I_{BC} \\ I_E &= I_{BE} + I_{CE} \\ I_{BE} &= \frac{I_{BF}}{\beta_F} + I_{LE} \\ I_{BC} &= \frac{I_{BR}}{\beta_R} + I_{LC} \\ I_{CE} &= I_{BF} - \frac{I_{BR}}{K_{QB}}, \end{aligned} \tag{21}$$

with

$$\begin{aligned} I_{BF} &= I_S\left(e^{\frac{V_{BE}}{N_F V_{th}}} - 1\right) \\ I_{LE} &= I_{SE}\left(e^{\frac{V_{BE}}{N_E V_{th}}} - 1\right) \\ I_{BR} &= I_S\left(e^{\frac{V_{BC}}{N_R V_{th}}} - 1\right) \\ I_{LC} &= I_{SEC}\left(e^{\frac{V_{BC}}{N_C V_{th}}} - 1\right), \end{aligned} \tag{22}$$

where I_S is the transport saturation current, N_F is the forward current emission coefficient, and I_{BF}, I_{LE}, I_{BR}, I_{LC}, β^*_F, $^*\beta_R$, and K_{QB} are the forward diffusion current, the non-ideal base-emitter current, the reverse diffusion current, the non-ideal base-collector current, the ideal maximum forward beta (BF in SPICE), the ideal maximum reverse beta (BR in SPICE), and the base charge factor, respectively.

The simulator has to solve a set of exponential functions by iterative approximation. Therefore, the optimizers considered in this work have to solve exponential problems, too.

The transconductance of a BJT is defined by:

$$g_m = \frac{I_C}{V_T}, \tag{23}$$

with I_C as the collector current at the Q-point, and V_T as the thermal voltage of the transistor.

Typical g_m values are in the range of 100–400 mS.

We use a replacement for the intrinsic spice3 NPN model for a real device, the transistor 2N3904. The transistor input and output transfer curves are shown in Figure 9. The plots show the high nonlinearity and saturation of the transistor model, which have a significant impact on the optimization algorithms evaluated in this work. The input and output transfer functions are totally different from the curves of OECT devices, as shown in Section 4.

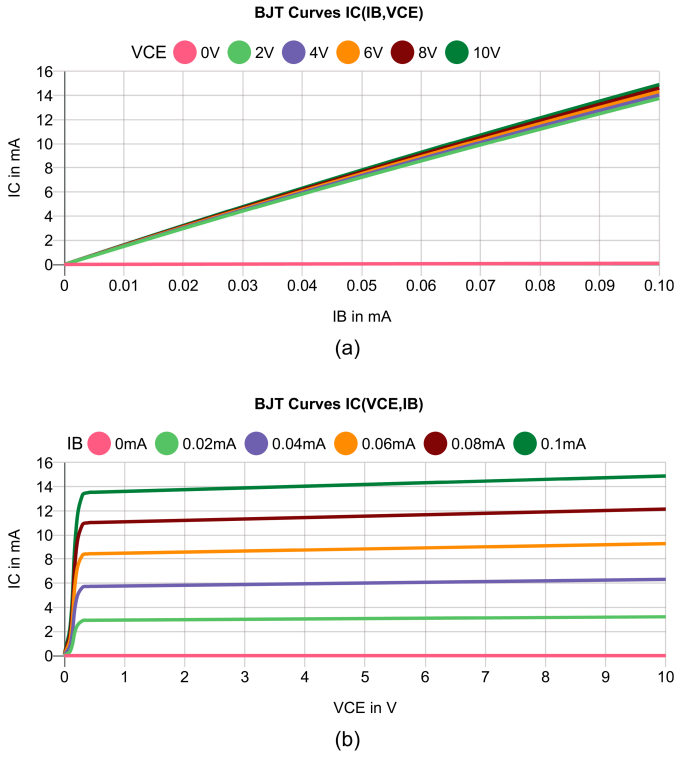

Figure 9. Examples of BJT transistor curves (spice BJT model with 2N3904 device parameter set). (**a**) IC(IB,VCE) (**b**) IC(VCE,IB).

The required control current reduces the decoupling among circuit stages, with an impact on the circuit optimization process (higher entanglement of component parameters), which can be improved by using FET devices, as discussed in the next Section.

3.2. FET

In addition to BJT devices, we deploy (junction) field-effect transistor (FET/JFET) devices, too. BJT devices are primarily current-controlled current sources, whereas FET devices are voltage-controlled current sources.

The transfer function of the JFET SPICE model is given according to Petrosyants et al. [38]:

$$I_{DS} = g_m(V_{GS} - V_{TO} - \gamma - V_{DS})^{VGEXP} I_{DS2},$$
$$I_{DS2} = (1 + \lambda V_{DS})\left(1 - \left(1 - \alpha \frac{V_{DS}}{3}\right)^3\right) \quad (24)$$

with I_{DS} as the drain-source current, V_{DS} is the voltage between the drain and the source of the transistor, g_m is the transconductance, V_{GS} is the gate-source control voltage, α is the saturation factor, V_{TO} is the threshold voltage, VGEXP is the gate voltage exponent, and λ is the channel length modulation coefficient.

Figure 10 shows exemplary JFET transistor curves, defining circuit voltage and current gradients, differing significantly from the BJT curves. The decrease in the drain current above a settling point (not visible in BJT devices) is important, which could have a significant impact on any loss- and gradient-based optimization.

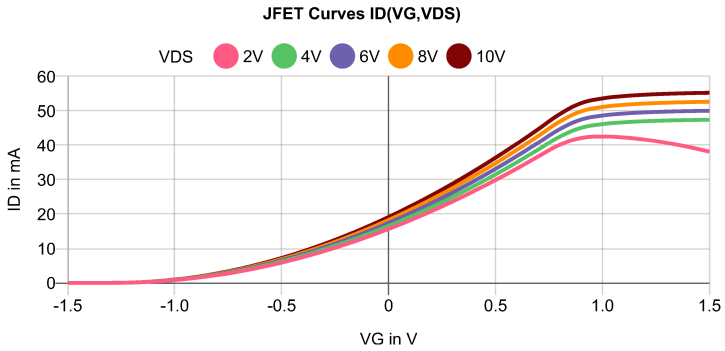

Figure 10. Examples of JFET transistor curves (spice JFET model with J113 device parameter set).

3.3. OECT

The organic electro-chemical transistor (OECT) is basically a voltage-controlled current source, with a physical model between a field-effect transistor (i.e., MOSFET, voltage-controlled) and current-controlled bipolar transistor principle (but not to be confused with OFET devices, which are based on the genuine field-effect principle). The gate current is much smaller than compared with BJT devices (multiple orders lower), but cannot be neglected compared with FET devices.

The transconductance of an OECT device depends on the geometrical parameters of the transistor, and is basically given by [39]:

$$g_m = \begin{cases} -\mu C \frac{Wd}{L} & V_D > V_G - V_T \\ -\mu C \frac{Wd}{L}(V_G - V_T) & V_D < V_G - V_T \end{cases}, \quad (25)$$

with C as the volumetric capacitance, V_g, V_d, and V_T as the gate, drain, and threshold voltages, respectively, and W, d, and L as geometric parameters.

Typical transconductance values g_m are in the range of 1–40 mS, much lower than the g_m values of BJT devices. The transconductance depends on the geometric layout of the transistor.

According to Sideris et al. [21], the drain current depending on the gate voltage is given by

$$I_D = \begin{cases} g_m\left(1 - \frac{V_{GS} - 0.5V_{DS}}{V_P}\right) & V_{DS} \geq V_{DS,sat} \\ -g_m\left(\frac{V_{DS,sat}^2}{2V_P}\right) & V_{DS} < V_{DS,sat} \end{cases} \quad (26)$$
$$V_{DS,sat} = V_{GS} - VP,$$

with V_P as the pinch-off voltage.

For electronic simulation, a numerical transistor model is required, giving the relationship between the drain and gate voltages and the drain current. The transfer function I(D,G) can be modeled by using a Verilog-A description, as shown in Algorithm 8. The $I(V_{DS}, V_{GS})$ function in the operational quadrant ($V_{DS} < 0$, $V_{GS} > 0$) was derived by symbolic regression (SR) from experimental measuring data, as discussed in Section 4. The other I(D,G) expressions were derived from physical models (TBA) with low accuracy and a significant reality gap, which is ultimately not relevant because the solution space of the optimization process must lie within the operational quadrant. A model that is valid in all quadrants, with smooth transitions of the transistor curve between quadrants, is required for convergence of the simulation itself, though.

Algorithm 8: An example of a Verilog-A OECT transistor model (derived from experimental data and physical modeling) for a horizontal, side-gated inkjet-printed OECT

```
module oect(g,d,s);
inout g,d,s;
electrical g,d,s;
parameter real Vp=2;    // Pinch-off voltage (V)
parameter real G=1e-5;  // Channel c o n d u c t a n c e (S)
//Variables
real Vgs,Vds,Vdsat,Ids;
analog begin
    Vgs=V(g,s);
    Vds=V(d,s);
    Vdsat=Vgs-Vp;
    // REGION CURRENTS
    if (Vds > 0) begin
      // A
      if ( -Vds <= Vdsat ) begin
        Ids = ( G * Vdsat * Vdsat ) / (2* Vp);
     end else begin
       Ids = -G *(1 -( Vgs -0.5* (-Vds) ) / Vp ) * (-Vds);
     end
    end else begin
     // uA
     if (Vgs < 0) begin
       Ids=((Vds / (exp(pow(0, 5)) + 3.5)) * 450)/1000000;
     end else begin
       Ids=((Vds / (exp(pow(Vgs, 5)) + 3.5)) * 450)/1000000;
     end
    end
    // A
    I(d,s)<+Ids;
end
endmodule
```

4. Experiments

The experimental section is divided into the optimization of a simple functional regression problem and three circuit optimization problems, using the ngspice simulator for the computation of the loss function. All electronic simulations were performed at room

temperature, T = 27 °C. The experiments were carried out using the JavaScript node.js platform (v20, 64 Bit Linux) using an embedded version of the *ngspice* simulator program (transpiled with the *emscripten* tool to WebAssembly). The optimization algorithms and the control loop were implemented in JavaScript, too. This monolithic approach requires no external software. Alternatively, but not used in this work, the ngspice software can be linked via a shared library by using the Foreign Function Interface (ffi).

There are two termination criteria for all optimizers:

1. The maximal number of iterations (epochs) is reached.
2. The loss is equal to or below the specified loss threshold. The loss threshold was chosen to produce solutions that satisfy the requested objectives, e.g., THD < 10%, or gain within tolerance of 5%. Because the loss threshold is a superposition of different losses, the threshold is an average solution quality measure.

If (1) but not (2) terminates the optimization, then the loss gradient converges to zero, too; i.e., there is no further progress, and the solutions are caught in local minima of the solution space, far away from solutions that satisfy the objectives.

The hyper-parameter space of the optimizer algorithms was constrained by selected suitable parameter sets derived from pre-experiments, which are:

1. GD: update rate $\alpha = 20 \in [1, 100]$ (adagrad), $\alpha = 2000 \in [1, 10{,}000]$ (gd standard), and $\alpha = 10 \in [1, 100]$ (Adam);
2. SA: cooling rate $0.999 \in [0.9, 0.99999]$ and randomness $0.05 \in [0.001, 0.1]$;
3. GA: randomness $0.3 \in [0.1, 0.9]$ and population size $30 \in [5, 50]$;
4. PSO: neighborhood size $10 \in [5, 30]$, ring neighborhood strategy, and population size $30 \in [5, 50]$.

4.1. Functional Regression

A functional polynomial regression

$$f(x) : \mathbb{R} \to \mathbb{R} = p_0 + p_1 x + p_2 x^2 + p_3 x^3 \tag{27}$$

is used as a simple standard model with the same parameter vector size as the circuit model. The functional regression is performed with a synthetic data table derived by the above function. The target parameter set is $p = [1, 2, -1.2, 0.5]$, with allowed ranges = $[[-2, 2], [-4, 4], [-2, 0], [-1, 1]]$.

The loss function for the functional optimization is given by:

$$l(y, y_0) = (y - y_0)^2, \tag{28}$$

applied to the reference data.

4.2. BJT and FET Devices

All optimization algorithms start with an initial randomized parameter set within the given suitable parameter ranges (uniform sampling). The optimization goal is the circuit component value estimation of the introduced single-transistor small-signal AC amplifier, with four resistors and the BJT model.

Due to the randomized parameter initialization, each optimization run will always produce a different result, regardless of the algorithm.

We chose the following target objective parameter values for the single transistor amplifier experiments:

1. Gain, $G = 5$;
2. Output impedance, $Z = 1000\ \Omega$;

3. Supply voltage, V_{cc} = 10 V (fixed);
4. Allowed static point collector voltage range [40%, 60%] of V_{cc};
5. Signal frequency 1000 Hz; signal amplitude V_s = 0.1 V.

For the OpAmp circuit, we chose the following target objective parameter values:

1. Gain G = 10 (programmed by external resistors);
2. Output impedance, Z = 200 Ω;
3. Supply voltage, V_{cc} = 10 V and V_{ee} = −10 V (fixed);
4. DC output voltage, V_{out} = 0 V;
5. Signal frequency 1000 Hz; signal amplitude V_s = 0.1 V.

Finally, for the sigmoid function approximation circuit, we had:

1. The scaled and stretched mathematical sigmoid function as the target objective (see Section 2.3);
2. Output range (0, 10 V);
3. Sigmoid function stretching factor w = 0.7;
4. Supply voltage V_{cc} = 10 V and V_{ee} = −10 V (fixed).

Polynomial function regression is used here to compare a medium nonlinear problem, defined by a smooth and dense gradient space, with the highly nonlinear "function" space of an electronic circuit without strict monotonic gradients, partial vanishing or steep gradient progression, occurrence of saturation (vanishing gradients, e.g., if the circuit operates outside of its static operating point), or a huge number of local minima in the sparse solution space.

We evaluate the three optimization methods with respect to:

1. Average number of iterations (epochs) and variance;
2. Average number of simulations;
3. The termination rate with an upper bound of iterations;
4. The parameter variance.

The loss function for the electronic circuit 1 (single transistor amplifier), as already introduced in Section 2.8, is given by:

$$\begin{aligned} l_a &= l_1 + \tanh(l_2) + l_3 \\ l_b &= l_1 + \tanh(l_2) + l_3 + l_4, \end{aligned} \tag{29}$$

with the goal of satisfying the given objective variables, such as the static DC operating point (l_1), gain (l_2), THD (l_3), and output impedance (l_4). The different objective variables are not independent. The loss function with four objective variables is the most challenging one because it narrows the solution space significantly and increases the chance of local minima traps, never reaching the requested loss threshold. The static operating point affects the output impedance, the gain affects THD, and so on.

A typical test program is shown in Example 1, using a parameterizable spice subcircuit. The test circuit as well as the simulation control loop are created by using constructor functions.

Figure 11 shows the simulated loss l_b (using electronic simulation), depending on the parameter values (resistors). The parameter space was explored by random sampling. Figure 11a,b show the parameter space with two selected resistor values and two remaining free parameters. The high loss noise is a result of the underdeterminacy by using two parameters only, and shows the difficulty of the circuit optimization. For example, the loss for specific R1–R2 values can result in a large variance with respect to the free parameters R3–R4. Figure 11c–f show a smooth parameter space, but only if the free parameters are fixed (here, to values from the valid circuits satisfying the objective functions).

Example 1. Computation of the loss function using embedded ngspice simulation (JS code).

```
// spuce signal amplifier circuit
var m = {
 circuit : [
 '.SUBCKT AMP1 VEE VCC IN OUT C R1VAL=1000 '+
       'R2VAL=1000 R3VAL=1000 R4VAL=1000',
 ' Q1 C 2 5 2N3904',
 ' R4 5 VEE {R4VAL}',
 ' R3 VCC C {R3VAL}',
 ' R1 VCC 2 {R1VAL}',
  ' R2 2 VEE {R2VAL}',
  ' C1 IN 2 100n',
  ' C2 C OUT 100n',
  '.ENDS'
  ].join('\n')
}

min=Math.min; max=Math.max
exp=Math.exp; tanh=Math.tanh

function loss(m,p) {
  // p is parameter vector with four resistor values
  var lib = spice.lib
  // spice test circuit using constructor functions
  m.test = [
    lib.source.dc('P','VCC',0,10),
    lib.source.dc('N','VEE',0,0),
    lib.source.sin('IN','IN',0,0.1,1000),
    lib.parameter('R1VAL',p[0]),
    lib.parameter('R2VAL',p[1]),
    lib.parameter('R3VAL',p[2]),
    lib.parameter('R4VAL',p[3]),
    lib.component.circuit(1,['VEE','VCC','IN','OUT','C'],
                  'AMP1',['R1VAL=R1VAL','R2VAL=R2VAL',
                  'R3VAL=R3VAL','R4VAL=R4VAL']),
    lib.component.resistor('L',['OUT',0],1E9)
   // mutable test resistor
  ]
  // constructors for ngspi<ce control loop (analysis)
  m.control = [
    lib.analysis.tran(2,'1u','10m',[]),
    lib.analysis.four(3,['IN','OUT'],1000),
     // THD analysis and gain
    lib.analysis.dc(4,['VEE','VCC','C']),
     // DC analysis
    lib.alter.resistor('L',1000),
     // Impedance analysis
```

Example 1. *Cont.*

```
43:    lib.analysis.tran(5,'1u','10m',[]),
44:    lib.analysis.four(6,['IN','OUT'],1000),
45:   ]
46:   // run spice
47:   var result =
         ng.probe([],lib.models,m.circuit,m.test,m.control),
48:      vc = result[2].voltages.c, // result from spice simulation
49:      thd = result[1].thd.out,
50:      gain =result[1].voltages.out[1].magnitude/
               result[1].voltages.in[1].magnitude,
51:      t = (result[4].voltages.out[1].magnitude/
              result[1].voltages.out[1].magnitude),
52:      outimp = 1000/t-1000
53:   var l1 = (1-exp(min(0,vc-vclow)))+(1-exp(min(0,vchigh-vc))),
54:       l2 = thd,
55:       l3 = abs(gain0-gain)/gain0,
56:       l4 = abs(outimp0-outimp)/outimp0,
57:       l = l1+tanh(l2)+l3+l4
58:   return l
59: }
```

4.3. OECT Devices

In this experiment, the single transistor amplifier is equipped with an ink-jet printed horizontal OECT using a data-driven OECT simulation model, differing significantly from the BJT model (wrt. the transistor curves and intrinsic transistor gain). The simulation model was derived from the experimental data. The transistors were manufactured to measure their transfer curves, ultimately used for creating an electronic simulation device model. The other circuit classes 2 and 3 are not considered here (due to a lack of n-channel complimentary transistors and models).

In contrast to the BJT circuit with an NPN transistor, this OECT circuit uses a p-channel transistor (for the sake of simplicity and available materials), which requires a negative supply voltage but a positive bias voltage. The maximal AC gain is lower compared to the BJT circuit due to lower transistor gain. Figure 12 shows the transistor layout of our pure PDOT:PSS OECT device with $CaCl_2$ liquid electrolyte (all layers printed), based on earlier work by Bosse et al. in [40], and Figure 13 shows three OECT devices printed on a glass substrate, using ink-jet printing technology (including electrolyte). Additionally, we used a data-driven OECT model from earlier work with a solid electrolyte and manufactured by classical lithography and deposition techniques (Au-iLCE-mPEDOT:PSS transistor, ref. [22]). We distinguish these two OECT devices by OECT-I and OECT-II labels, respectively.

The OECT-I transistors were manufactured with a Fujifilm Dimatix DMP-2831 drop-on-demand inkjet printer with a DMC-11610 print head, with sixteen 20 µm nozzles and about 10 pl droplet volumes. The organic conductor material was PEDOT:PSS, with ethylene glycol as an additive improving conductivity and viscosity; the liquid electrolyte was 6 wt% $CaCl_2$ dissolved in deionized water. The substrate (for experimental studies) was glass, with a thickness of 0.7 mm. The organic conductor was used for the transistor channel (low thickness and monolayer), as well as for the contact electrodes (higher thickness and five printed layers). Printing was completed with one nozzle and a drop spacing of 20 µm. The

substrate plate was heated to 60 °C, and the distance between the nozzles and the glass substrate was about 1 mm.

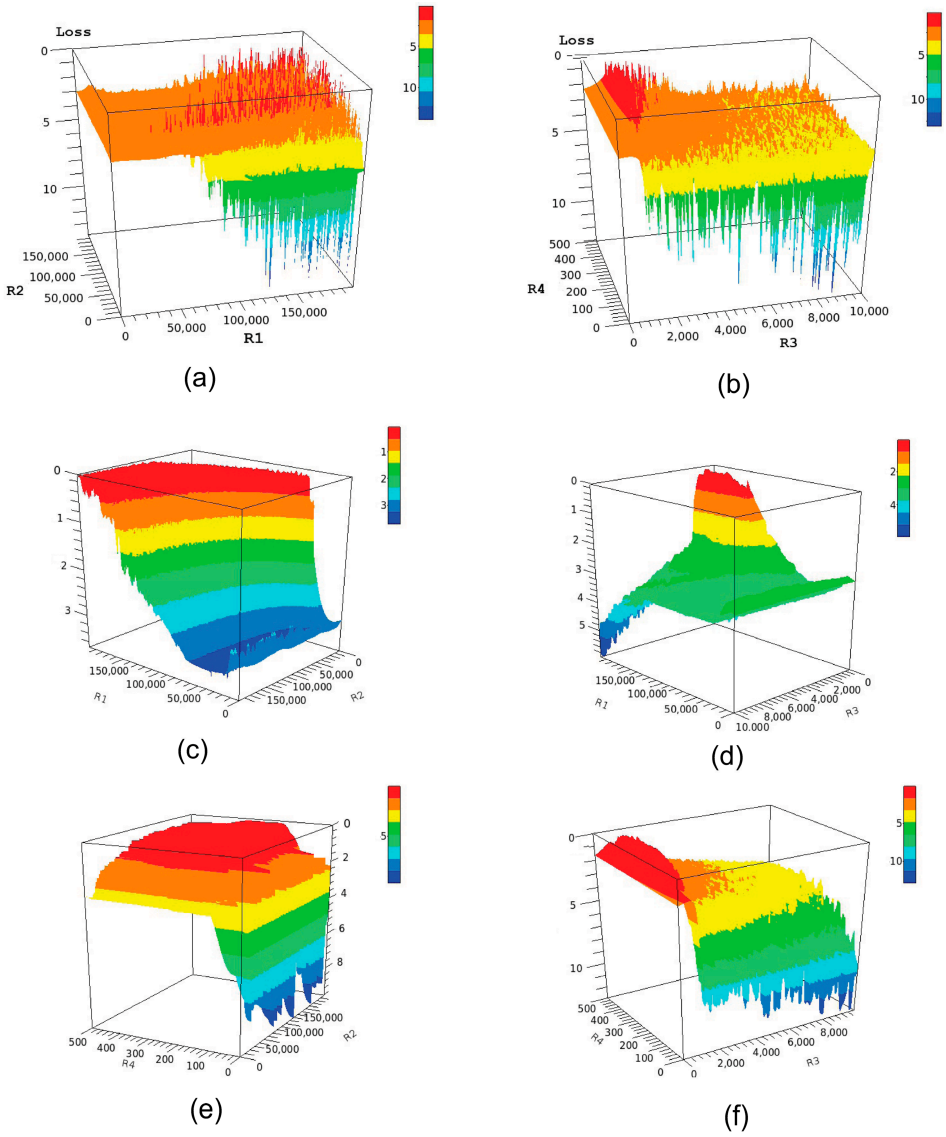

Figure 11. Some insights into the parameter space of the BJT amplifier. All resistor values (x- and y-axis of plots) in Ω units. (**a**) The loss l_b | R1–R2 plot, (**b**) the loss l_b | R3-R4 plot, (**c**) the loss l_b | R1–R2 plot with fixed R3–R4, (**d**) the loss l_b | R1–R3 plot with fixed R2–R4, (**e**) the loss l_b | R2–R4 plot with fixed R1–R3, and (**f**) the loss l_b | R3–R4 plot with fixed R1–R2.

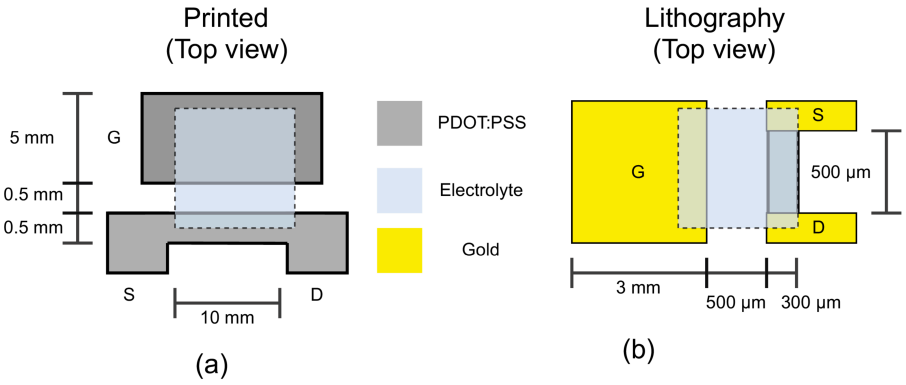

Figure 12. The OECT transistor layout. (**a**) The metal-free PDOT:PSS OECT with the electrolyte $CaCl_2$, and (**b**) the Au-iLCE-mPEDOT:PSS OECT [22].

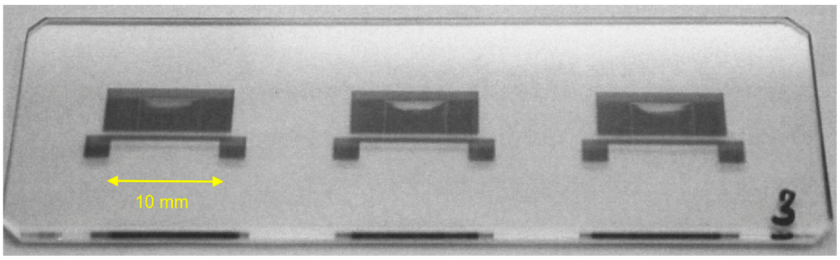

Figure 13. Three OECT devices printed on glass substrate using ink-jet printing technology (including electrolyte) (photo by J. van Kempten).

Figure 14 compares the transistor curves $I_D(V_{DS}, V_{GS})$ of our horizontal PEDOT:PSS and an older horizontal Au-PEDOT:PSS OECT [22]. The transistor curves of the pure PEDOT:PSS (side-gated) OECT are closer to the BJT/FET curves than the curves of the mixed Au-PEDOT:PSS OECT devices from Bornemann et al. [22], which is caused by a different layout, too. We optimized the circuits using both device architectures. The maximal gain of a single OECT transistor amplifier is 3.5 and 2, for the PEDOT:PSS and Au-PEDOT:PSS transistors, respectively. The Au-PEDOT:PSS OECT shows early saturation for $V_{DS} < -0.5$ V.

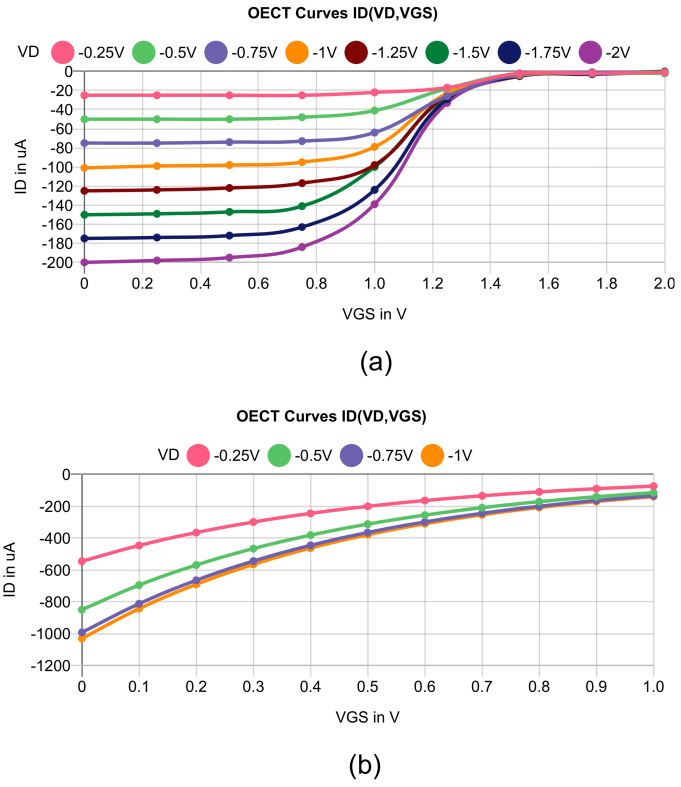

Figure 14. Comparison of transistor curves for (**a**) pure PEDOT:PSS horizontal and (**b**) Au-PEDOT:PSS horizontal layout OECT devices (OECT data from [22]).

The electronic simulation using the pre-compiled OSDI OECT model is up to 10 times slower (using the native *ngspice* version with *osdi* support) than using the built-in BJT models, although the native *ngspice* version exploits control path parallelism. Due to the low intrinsic transistor gain of the OECT devices, we lowered the target gain of the AC amplifier to $G = 2$–5.

We chose the following target objective parameter values for the OECT amplifier experiments, reflecting different transistor behavior and reduced transconductance of OECT devices compared with their BJT counterpart:

1. Gain G = 5 (OECT-I) and G = 2 (OECT-II);
2. No output impedance constraint (due to limited and narrow parameter space compared with BJT);
3. Supply voltage, V_{dd} = −4 V (OECT-I) and −2 V (OECT-II), and bias voltage, V_b = 2 V;
4. Allowed static point drain voltage range [15%, 85%] of V_{dd};
5. Signal frequency 1000 Hz, and signal amplitude, V_s = 0.1 V;
6. Loss l_2 of THD was scaled by a factor of 1/20 to reflect reduced linearity of OECT versus BJT transfer curves.

The lower gain (2) setting was used for the OECT-II circuit and is a result of the low transconductance of the OECT-II device in the linear part of the transistor curve.

4.4. Advanced Circuit Analysis

The previous experiments focused on the comparison of all optimization algorithms and the single transistor circuit, using BJT and OECT devices. The PSO algorithm outperformed all other algorithms in terms of success rate (reaching the optimization goal within a given number of iterations) and the total number of required simulations.

This section focuses on the more complex circuits 2 (discrete OpAmp) and 3 (sigmoid function), both consisting of three transistors and eight parameterizable components (resistors).

The loss function for circuit 2 (discrete OpAmp), as already introduced in Section 2.8, is given by:

$$\begin{aligned} l_a &= l_1 + \tanh(l_2) + l_3 \\ l_b &= l_1 + \tanh(l_2) + l_3 + l_4, \end{aligned} \tag{30}$$

with the goal of again satisfying the given objective variables: the static DC operating point (l_1), gain (l_2), THD (l_3), and output impedance (l_4). In contrast to circuit 1, the DC operating point is the output voltage with zero difference input voltage. Additionally, the gain is programmable by two resistors, but is influenced by the other component values, too.

The loss function for the electronic circuit 3 (sigmoid function) relies on a functional model loss and is given by:

$$\begin{aligned} l_1 &= |V_{\text{out}}| \\ l_2 &= \sqrt{\sum_i (sigmoid(V_{\text{in,i}}) - V_{\text{out,i}})^2} \\ l &= \tanh(l_1) + l_2 \end{aligned} \tag{31}$$

for the requested input (x) range V_{in} = [−10 V, 10 V].

5. Results

The computation time of the functional model was negligible, but the computation time of the electronic model, i.e., by running the electronic simulation via embedded or external *ngspice*, required about 100 ms on a typical desktop computer (Intel Core i7-7600U CPU @ 2.80GHz), which creates significant optimization times. Although, the compiled WebAsm *ngspice* version introduced computational overhead, the call of the native ngspice program, including file transfers, required 30% more time (at least for this simple simulation model). Using an externally imported OSDI device module increased the computation time by another factor of 10.

We considered experiments with the less restricted l_a loss function (with DC, THD, and gain objectives) and the more restricted l_b loss function (with DC, THD, gain, and impedance objectives). We analyzed the results of the four optimization algorithms GD,

SA, GA, and PSO. The standard GD algorithm was additionally compared with two modifications, the Adam and AdaGrad optimizers.

The average termination rates (successful optimization) are shown in Table 3.

Table 3. Average termination rates for circuit 1 optimization (successful optimization, n = 100 experiments for each algorithm).

Algorithm	Polynomial (Loss MSE)	Circuit BJT (Loss l_a)	Circuit BJT (Loss l_b)	Circuit OECT-I (Loss l_a)	Circuit OECT-II (Loss l_a)
GD	100%	63%	34%	0%	3%
GD-AdaGrad	100%	38%	15%	1%	8%
GD-Adam	100%	58%	18%	26%	30%
SA	100%	72%	46%	100%	100%
GA	100%	83%	33%	97%	92%
PSO	100%	100%	100%	100%	100%

The termination rates depend on the loss function. A more constrained loss l_b decreases the probability of successful optimization, for some algorithms significantly. The GD algorithm has the lowest termination rate for the circuit optimization. The successful optimization depends here mainly on the random parameter initialization and the circuit with its transistor model, which shows high nonlinearity, non-monotonic gradients of the loss function, and a high number of local minima spread in the parameter space. The GD-Adam and GD-AdaGrad algorithms perform significantly better with OECT models, and Adam optimizes the OECT circuits with acceptable rates much better than AdaGrad and standard GD. Variations in the optimizer parameters (e.g., update and decay rates) do not alter the results significantly.

The SA and GA algorithms show a suitable convergence rate for the optimization without impedance constraints, but degrade significantly for the optimization with the impedance constraint. Only the PSO algorithm converges with a rate of 100% for all optimization scenarios (numerical function, BJT, and OECT circuits) with a low number of epochs. Surprisingly, SA and GAs perform well on the OECT circuits with high termination rates.

Figure 15 (left side) shows the number of required epochs to terminate with a given loss threshold (0.1 BJT, 0.15 OECT circuit) and a specific loss function, l_a or l_b. The computational complexity of the optimization process is mainly given by the electronic simulator, as shown on the right side of Figure 15. The number of simulations required for each epoch varies strongly with the different optimization algorithms and relies partly on the size of the parameter vector. Figure 16 shows the results for the OECT-I and OECT-II amplifier circuits, respectively. SA still shows the highest number of epochs, but GD-ADAM shows the highest number of simulations. The PSO algorithm requires the lowest number of epochs and simulations.

The experiments showed that the GD optimization is stable using the polynomial regression model, but significantly unstable with the electronic circuit model, i.e., by showing a high number of non-terminated runs and a high average number of iterations. The main reasons are the high nonlinearity of the model due to the BJT (or OECT) and the very low gradient if the circuit is outside of its static operating area (saturated circuit).

GD and SA start with an initial random parameter set and optimize only one solution; GAs and PSO create a population of solution candidates, which are combined during optimization. Although SA has mostly a high termination rate, a high average number of epochs is required (about 1500 for the BJT circuit). GAs and PSO converge much faster, even under consideration of the required simulations below the average required for SA, but depending on the population size.

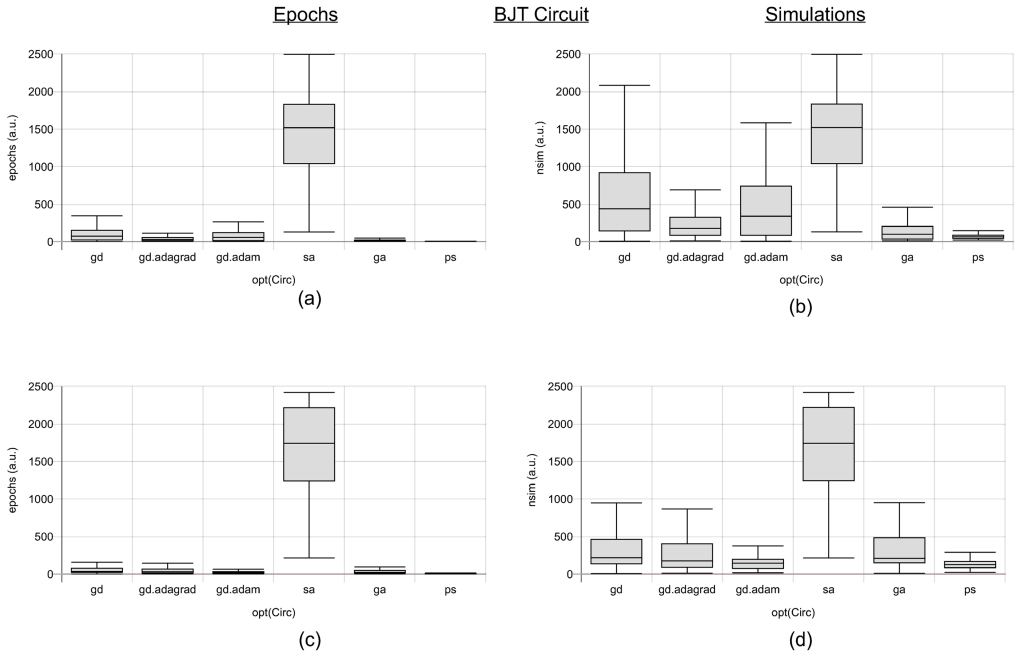

Figure 15. A comparison of the average number of epochs (**a**,**c**), as well as the number of simulations (**b**,**d**), and their variance required for termination with a loss = 0.1 for the different optimization algorithms performing parameter optimization of the BJT amplifier circuit 1: (**a**,**b**) l_a = loss (DC, gain, and THD); (**c**,**d**) l_b = loss(DC, gain, THD, and impedance).

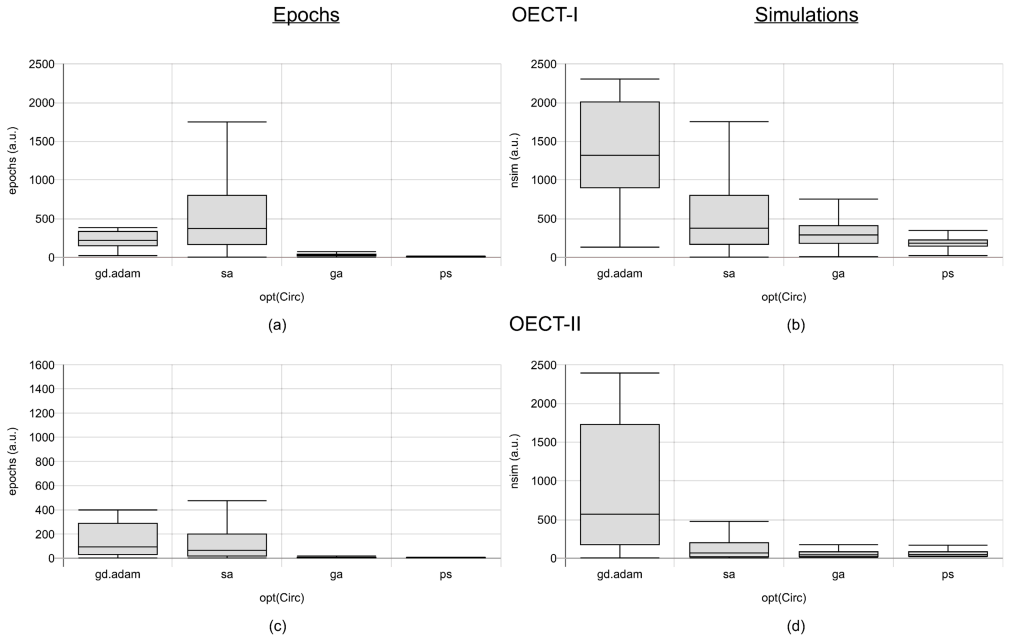

Figure 16. A comparison of the average number of epochs (**a**,**c**), as well as the number of simulations (**b**,**d**), and their variance required for termination with a loss = 0.1 for the different optimization algorithms performing parameter optimization of the OECT-I (**a**,**b**) and OECT-II (**c**,**d**) amplifier circuit 1 with l_a = loss (DC, gain, and THD).

Finally, PSO outperforms all other algorithms in terms of speed and termination rates, with respect to our experiments.

Figures 17 and 18 show the parameter variance in the BJT circuit and OECT circuit optimization, respectively. Figure 17 compares the results for the l_a and l_b loss functions, too. All optimizers show a larger but comparable variance in the resistor values R_1 and R_2, determining the static operating point of the amplifier, except for SA in the BJT circuit,

independent on the loss function (without and with output impedance constraints). The results always differ significantly from the originally calculated values (shown by dotted lines) using electrical laws and transistor curve quiescent point analysis.

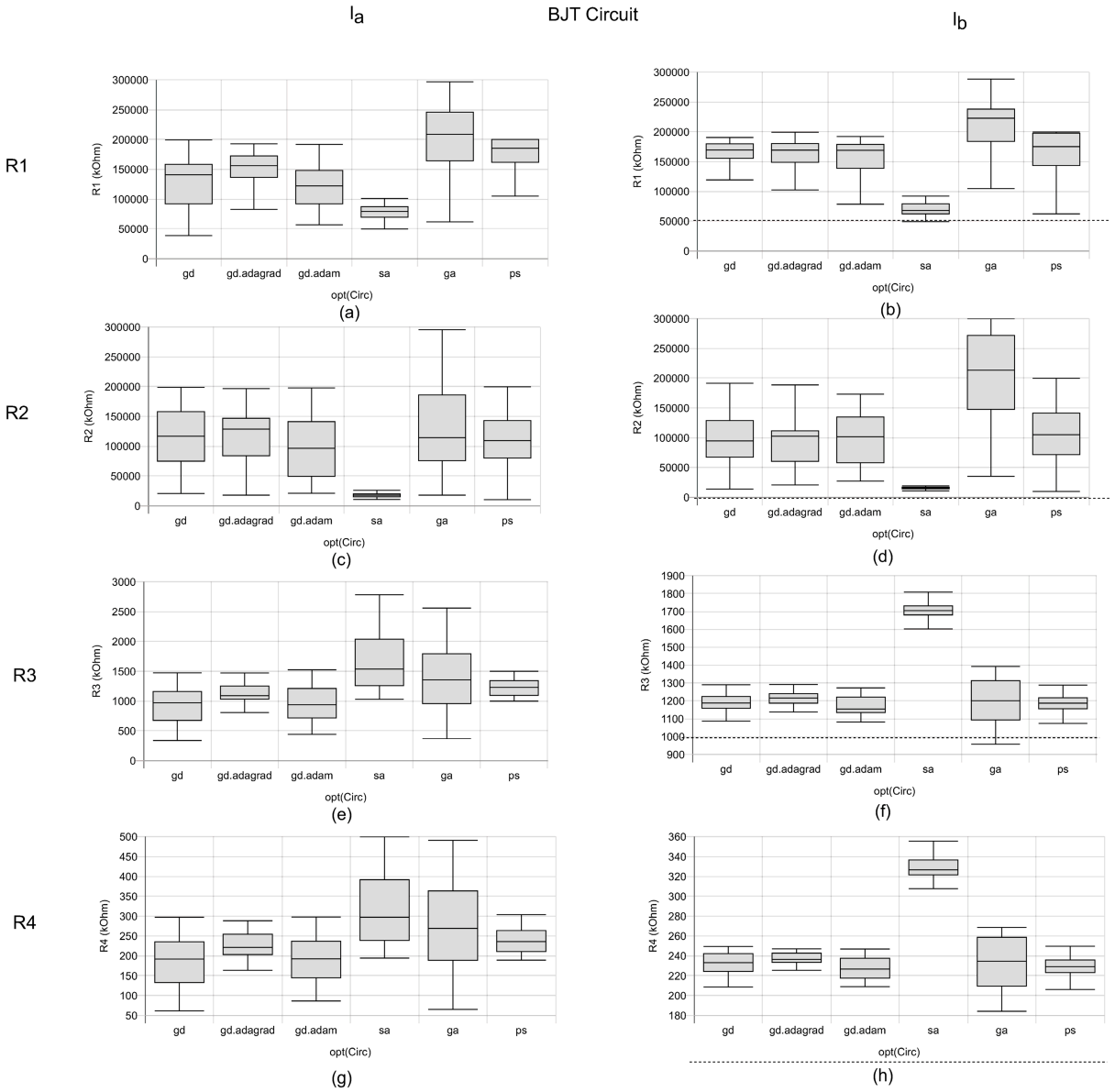

Figure 17. A comparison of the circuit component parameter variance for the different optimization algorithms applied to the BJT amplifier circuit. The reference values from the electrical calculations are R_1 = 55 kΩ and R_3 = 1 kΩ, as shown by the dotted lines. (**a**,**c**,**e**,**g**) Loss(DC, gain, THD), and (**b**,**d**,**f**,**h**) loss (DC, gain, THD, impedance).

For the resistors R_3 and R_4, determining the gain of the amplifier, the variance is large for all optimizers without output impedance constraint, and smaller variance with output impedance constraint, except for the GA case. The OECT optimizations are always carried out without an output impedance constraint. The variances in R_1 and R_2 are still high, whereas the variances in R_3 and R_4 are low for OECT-I and all stochastic algorithms. For the OECT-II amplifier, only R_4 shows slow variance; R_3 always shows a high variance. These results demonstrate the strong dependency of the circuit parameter optimization on the transistor models.

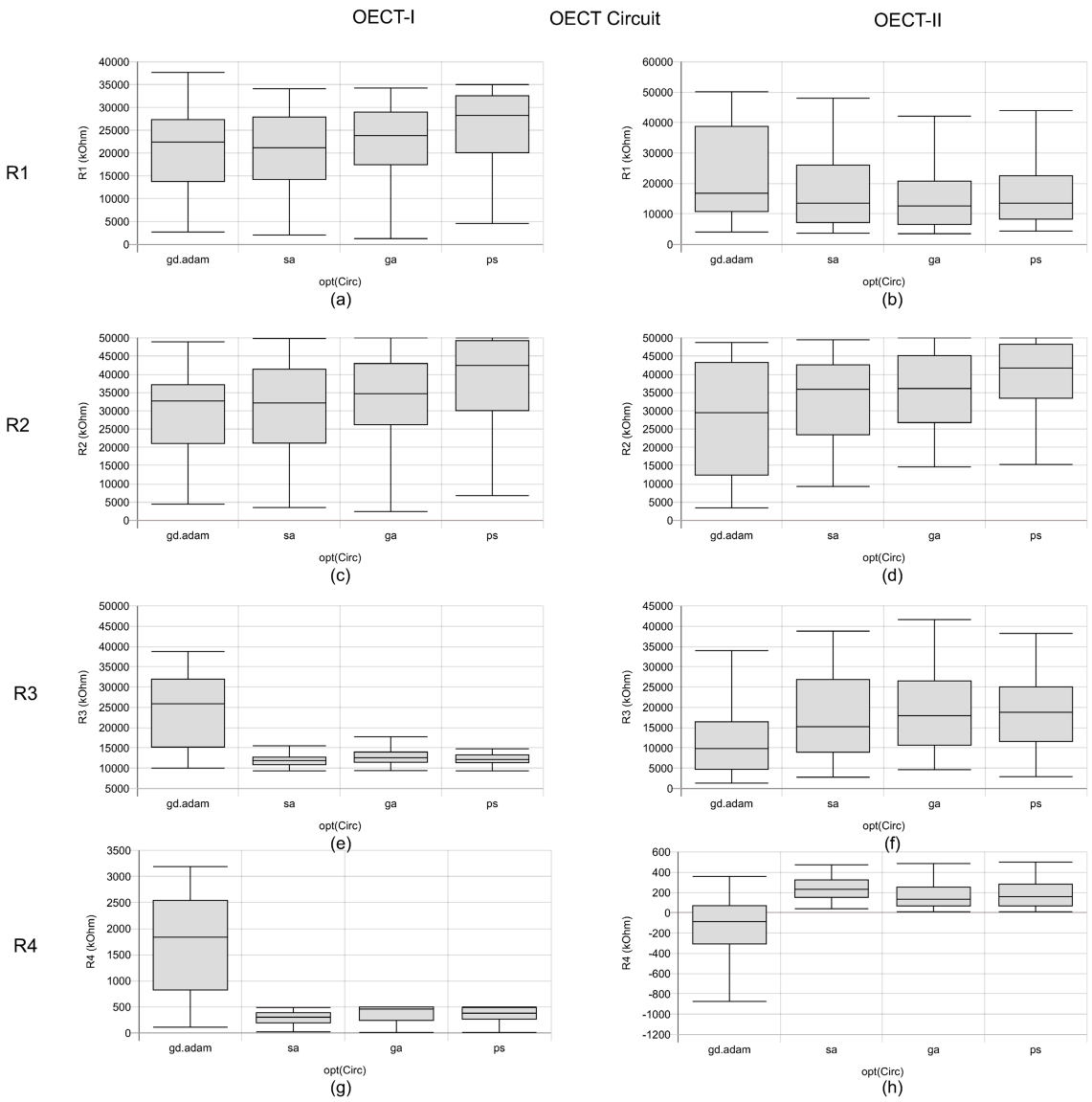

Figure 18. A comparison of the circuit component parameter variance for the different optimization algorithms applied to the OCET amplifier circuits (Loss (DC, gain, and THD). (**a**,**c**,**e**,**g**) OECT-I device; (**b**,**d**,**f**,**h**) OECT-II device.

The parameter solution space is constrained by the range intervals of the parameters (i.e., the resistor values). But using hard constraints on the parameter space can result in zero gradients, which stops any non-stochastic algorithm like GD, given an additional explanation of the low termination rates.

To summarize: Population-based stochastic optimization methods (GAs and PSO) are superior compared with single-instance methods (GD and SA). They provide stable convergence, a lower number of epochs and, hence, required simulations, and suitable parameter solutions. GD is not suitable for the optimization of multivariate, multi-objective, highly nonlinear functions, and even SA does not perform well and shows low convergence rates. Additionally, the number of time-critical simulations depends on the size of the parameter vector. Pareto-driven optimization is difficult to achieve with such a model function to be optimized. The PSO algorithm outperformed all other algorithms. The optimization depends on the transistor model (e.g., transconductance, transistor curves, and so on).

Although GA is primarily intended for discrete and combinatorial problems, the modified GA version for solving continuous problems can compete with SA and PSO, intended for continuous problems.

Finally, we analyzed the circuit behavior under parameter variation using Monte Carlo simulation. We processed all computed solutions from all algorithms and varied the parameters within a 20% tolerance (100 samples). The objective's gain, distortion, and output impedance were measured by simulation, as shown in Figure 19. For the BJT circuit, the solutions of SA show higher variance in the objectives and sensitivity to parameter variation, indicating slightly lower solution quality. In contrast to the OECT circuits, the solutions from GD-Adam show the highest sensitivity to parameter variations. But there is no unified conclusion as to which optimizer provides statistically better solutions in terms of parameter variation, which is technically of high relevance.

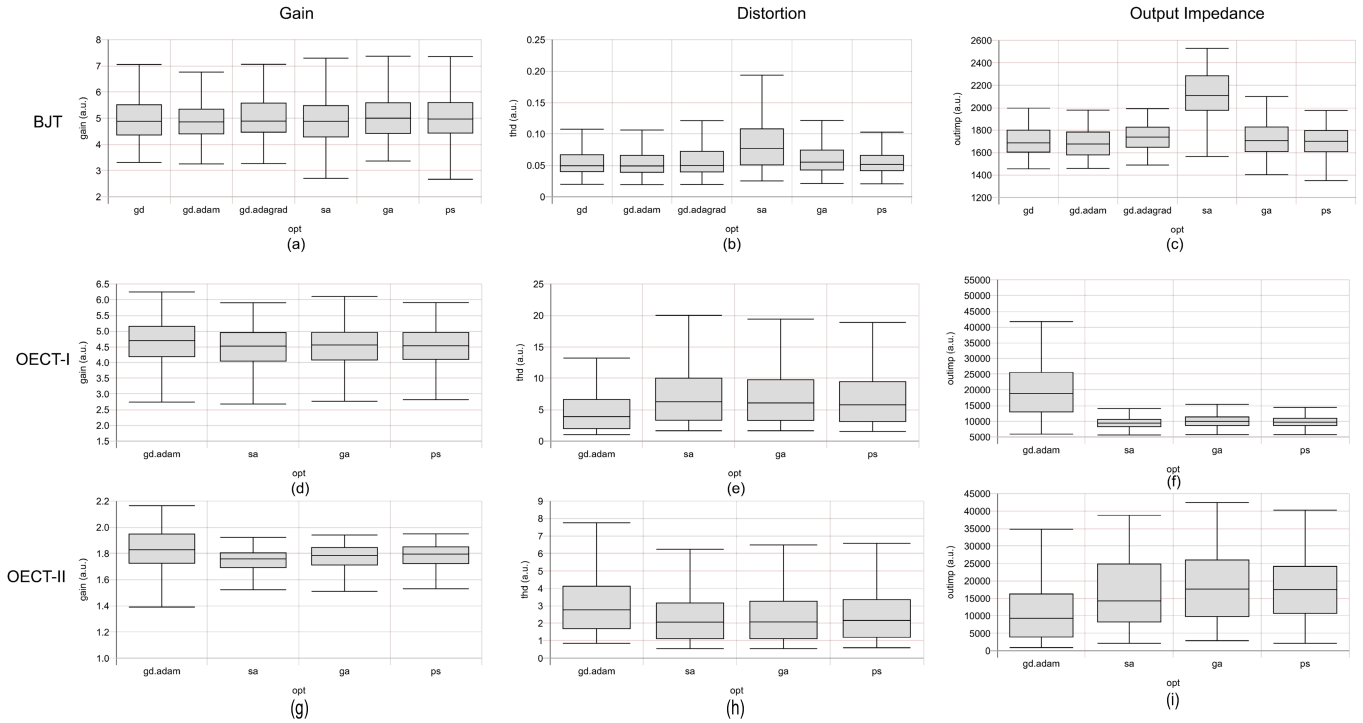

Figure 19. The results for the Monte Carlo simulation of circuit parameters by applying a 20% tolerance to all solutions computed by the optimizers. (**a**,**d**,**g**): Gain, (**b**,**e**,**h**): distortion, and (**c**,**f**,**i**): output impedance. (**a**–**c**): BJT, (**d**–**f**): OECT-I, and (**g**–**i**): OECT-II.

Figure 20 shows statistics for the optimization of circuit 2 (discrete OpAmp), and Figure 21 for the functional circuit 3 (sigmoid function approximation), both optimizing eight parameters, both with three nonlinear elements. PSO always succeeds, although in the functional approximation circuit, the termination rate drops to 40%. GD (with all its variants) fails in both circuits (neither termination within the given number of iterations nor any suitable result). SA fails completely in circuit 3.

PSO (with a population size of 30) requires less than 30 epochs on average to find a solution. Although SA, with a single solution candidate, requires many more iterations, the number of required and time-critical simulations is in the same order. Even with more complex circuits and a function approximation, the number of epochs is comparable to that of the single-transistor circuit. Finally, Figure 22 shows the parameter variation in the PSO algorithm for circuits 2 and 3. The stochastic optimization shows for both circuits a low variation in resistor values, except R_4 in circuit 4, which has no relevance for the circuits design driving the emitter follower at the output.

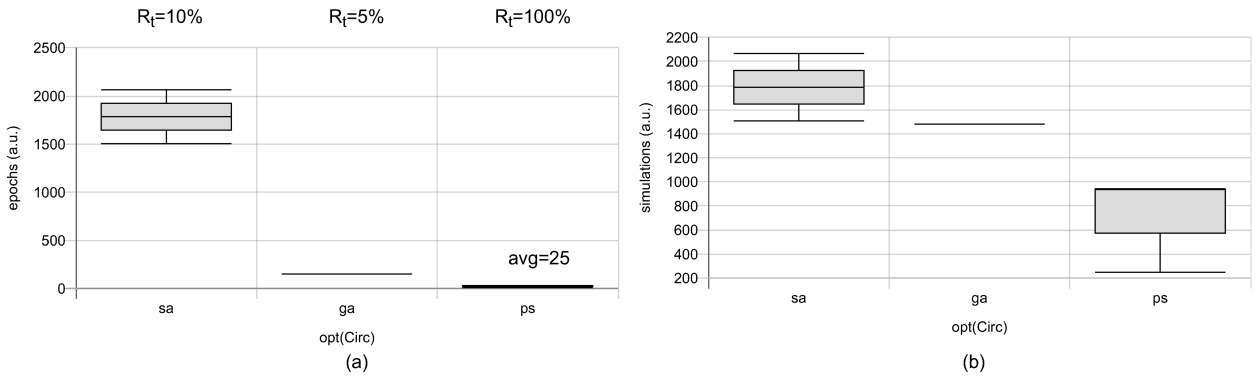

Figure 20. The statistics for the epochs (**a**) and simulations (**b**) for the circuit 2 (discrete OpAmp) optimization. Only PSO terminated with a rate of R_t = 100%. Shown are the statistics only for the terminated runs reaching the loss threshold (0.1) with an original N = 100 experiments. GD failed completely.

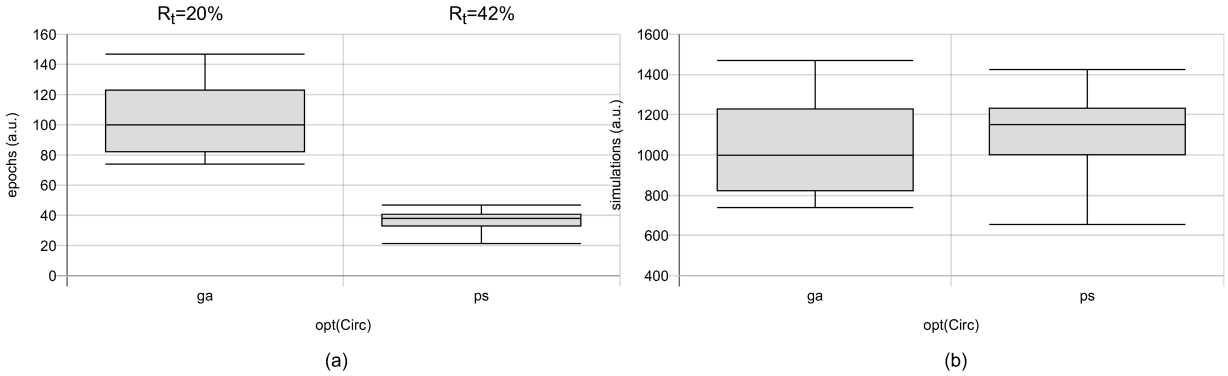

Figure 21. The statistics for the epochs (**a**) and simulations (**b**) for the circuit 3 (sigmoid function) optimization. Shown are the statistics only for terminated runs reaching the loss threshold (0.45), with the original N = 100 experiments. GD and SA failed completely.

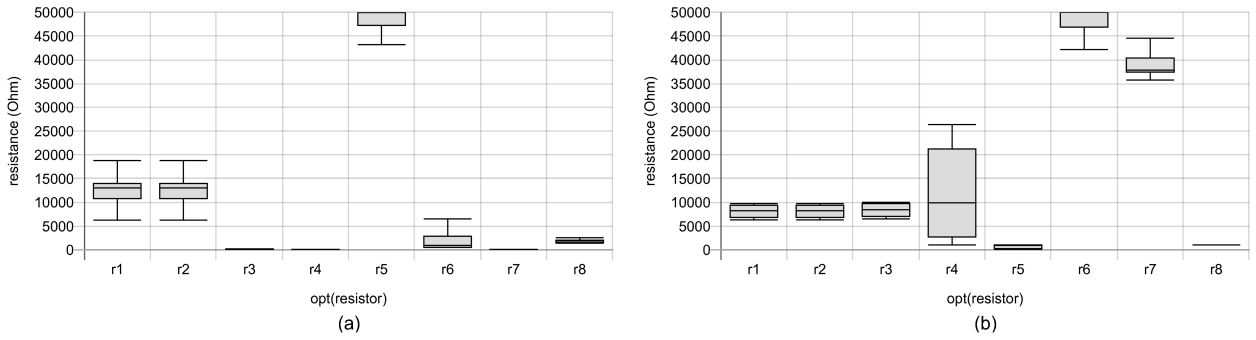

Figure 22. The statistics for the parameter (component value) variations in the PSO results for circuit 2 (**a**) and 3 (**b**), each with 8 parameters.

6. Conclusions and Outlook

In this work, the suitability of different optimization algorithms deployed for the parametric search of circuit component values (resistor values) using different transistor models (BJT, FET, and OECT) was evaluated. The optimizers had to solve a multivariate and multi-objective search problem. Although a very simple circuit of a signal amplifier was used initially, the experiments identified the behavior and the limitations of the optimization methods. The methods were applied to two more complex circuits, showing similar results. We found that PSO outperforms all other algorithms in speed and termination

rates (convergence) with respect to our experiments and the chosen hyper-parameter sets of the optimizers, which were derived by pre-experiments in advance. A typical PSO run requires about 100 simulations for the simple amplifier circuit, about 500 for the discrete OpAmp, and about 1000 simulations for the sigmoid function circuit.

In contrast to data-driven numerical regression, circuit optimization uses electronic simulation-in-the-loop to compute the loss function, which is an aggregate function composed of different objectives, i.e., the static DC operating point, the gain of the amplifier, the signal distortion, and optionally the output impedance. To narrow the search space, parameter constraints were introduced, too. Too strict as well as too wide parameter ranges have a negative impact on optimization convergence. The parameter ranges should be set carefully by iterative adaptation. Table 4 summarizes the advantages and disadvantages of the different optimization methods used in this work.

Table 4. Summary of advantages and disadvantages of different optimization methods used in this work.

Method	Advantages	Disadvantages
Gradient Descent	Numerical explainable algorithm.	Impact of initial random parameter initialization is high, a differentiable model without poles is required, otherwise vanishing or exploding gradients prevent termination. Numerical gradient computation for circuits is expensive requiring a high number of simulations and depends on the number of parameters. Inherently limited to numerical problems.
Simulated Annealing	Simple pure stochastic algorithm. Only one simulation required in each iteration.	No assessment of solution progress (only better or worser or random enforcement of solution acceptance). Weak termination rate for circuit optimization.
Genetic Algorithm	Population-based algorithm with combination of solutions. Simple encoding of structure, with modification simple encoding of numerical values.	No assessment of solution progress (only relational). Medium termination rates for circuit optimization.
Particle Swarm Optimization	High termination rate, population-based, assessment of solution progress, combination of solutions.	Initially limited to numerical optimization problems, extension to binary structural problems has impact on solution progress evaluation.

We found that the convergence of the compared algorithms depends on the constraints (with and without output impedance), the initial seeding of the parameter vector (except for PSO, and partially for GA), and the transistor device (BJT vs. FET vs. OECT). The BJT has a much higher transconductance and nonlinearity than the OECT, having a significant impact on the convergence (if ever) of the parameter search or computation (GD). Stochastic algorithms with randomness are better suited for parameter search in a highly sparse but widespread parameter space, with a large number of local loss minima. Vanishing gradients, typical for transistor circuits in saturation and far away from the operating point, prohibit successful optimization using gradient-based algorithms, especially with parameter constraints. Population-based approaches (GAs and PSO) with multiple solutions evaluated in parallel outperform single-solution approaches (GD and SA).

One significant factor in parameter optimization of electronic circuits is the computational time, mainly given by the electronic simulator (with a typical simulation time of

about 100 ms for a circuit with less than 10 transistors). The average number of required epochs of the algorithms differs significantly, but the number of required simulations is relevant. GD required $|P| + 2$, SA 1, GA, as well as PSO, $|pop|$ simulations per epoch, with $|P|$ as the parameter vector size and $|pop|$ the population size.

All solutions of successful optimization satisfy the given objectives; therefore, all solutions can be considered of the same quality. But in circuit design there are many more effects that can compromise the circuit behavior, e.g., sensitivity to temperature or component value variations. We tested the solutions from all optimizers for their sensitivity to parameter variation using Monte Carlo simulation, but found no significant difference, except some circuit-optimizer-specific artifacts, difficult to be explained and foreseen.

We showed with our experiments that the transistor physics (with respect to the transistor curves by comparing BJT, FET, and two different OECT devices) has a significant impact on the optimization speed and termination rates. Stochastic optimization algorithms can find a broad parameter solution space, with circuits all satisfying the given constraints. But the more complex and constrained circuits 2 and 3 showed low parameter variations, although all solutions were seeded randomly. All algorithms were constrained by parameter range limits, which must be chosen carefully to avoid a too large search space and to avoid solutions that cannot be further optimized by too strict parameter ranges (lowering the termination rate).

The GA was originally intended for solving structural/combinatoric problems only. We modified the algorithm for continuous problems by modifying the bit-like genome vector to a numerical vector with continuous mutation and linear crossover. We got reasonable results with this approach. But, PSO was the only algorithm that delivered results independent from the transistor device (and model) with a 100% convergence rate within given constraints and limits, regardless of the transistor device and their elect4ronic models, requiring the lowest number of epochs and total number of simulations. Only the functional approximation circuit with an MSE loss function showed a degradation of the PSO termination rate down to about 40%, but significantly higher than the other algorithms (GD and mostly SA failed in 100% of runs in circuit 2 and 3 optimization).

As a next step the found solutions from the different optimizers should be evaluated with respect to temperature or component value variations using much more complex circuits. Finally, the continuous parametric optimization methods should be combined with discrete combinatorial optimizers, changing the structure of the circuit, e.g., to improve reliability and reducing sensitivity to temperature and component variations, or to find the minimum transistor count for a circuit with given constraints. Organic electronic components, especially, show accelerated altering and hysteresis effects, which should be incorporated into the simulation and optimization process.

The optimization of an amplifier is a closed problem, i.e., the structure is given and fixed. In contrast, the optimization of an analog artificial neural network for classification and data regression, for example, is an open problem, where the structure (the number of neurons) is a free structural parameter, too, making the optimization process much more complicated. Finding the Pareto-optimal solutions has still not been solved and will be added in the future as an additional objective (loss function).

Altering and hysteresis are relevant in OECT devices and will have significant impact on circuit optimization (the device changes during operation, similar to heating), which introduces, basically, a device parameter state with history. But that would go beyond the scope of this article; we focused on the investigation of different optimization algorithms. In addition to Monte Carlo simulation, we will investigate the altering and hysteresis effects, and extend the electronic device simulation model with at least a simple deviation model controlled by the main simulation control loop (spice itself does not support parameter

state change based on history). Finally, the extension of the optimization to the multi-scale level, e.g., by including geometric transistor layout parameters, will increase the benefit of the presented work for OECT circuit designs significantly.

Author Contributions: Conceptualization, S.B.; methodology, S.B. and S.K.; software, S.B.; validation, S.B. and S.K.; formal analysis, S.B.; investigation, S.B. and S.K.; resources, S.B. and S.K.; data curation, S.B.; writing—original draft preparation, S.B. and S.K.; writing—review and editing, S.B. and S.K.; visualization, S.B. and S.K.; supervision, S.B.; project administration, S.B.; funding acquisition, S.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data is available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

```
// random topology of informers
function CommInitRandom (comm, Ψ)
 comm = 0
 ∀i ∈ {1,2,..,Ψ.size} do
  comm [i,i] := 1 // diagonal
  ∀k ∈ {1,2,..,Ψ.n_ng} do
   j = runif(1,Ψ.size)
   // particle i informs particle j
   comm[i,j] := 1
  done
 done
end
// global neighborhood
function InformGlobal (pos_nb,g_best,Ψ)
 // all particles have the same attractor (gbest)
 // copy the contents of gbest to pos_nb
 ∀i ∈ {1,2,..,Ψ.size} do
  pos_nb[i]=clone(g_best)
 done
end
function InformRandom(comm,pos_nb,pos_n,fit_b,improved,Ψ)
 if not improved then CommInitRandom(comm,Ψ) end
 Inform(comm, pos_nb,pos_b,fit_b,Ψ)
end
```

References

1. Zadeh, D.N.; Elamien, M.H. Generative AI for Analog Integrated Circuit Design: Methodologies and Applications. *IEEE Access* **2024**, *13*, 58043–58059. [CrossRef]
2. Mina, R.; Jabbour, C.; Sakr, G.E. A review of machine learning techniques in analog integrated circuit design automation. *Electronics* **2022**, *11*, 435. [CrossRef]
3. Dutta, R.; James, A.; Raju, S.; Jeon, Y.-J.; Foo, C.S.; Tshun Chuan Chai, K. Automated Deep Learning Platform for Accelerated Analog Circuit Design. In Proceedings of the 2022 IEEE 35th International System-on-Chip Conference (SOCC), Belfast, UK, 5–8 September 2022; pp. 1–5. [CrossRef]
4. Granelli, R.; Demartis, V.M.; Frusconi, G.; Kovács-Vajna, Z.M.; Torricelli, F. Organic Electrochemical Transistors Monolithically Integrated with Precise Micro-Dispensing Enable High-Performance Biosignals Amplification. *Adv. Sci.* **2025**, *12*, e08872. [CrossRef] [PubMed]

5. Greenwood, G.W.; Tyrrell, A.M. *Introduction To Evolvable Hardware: A Practical Guide for Designing Self-Adaptive Systems*; IEEE Press: New York, NY, USA, 2007.
6. Xu, W.-J.; Xin, D.-J.; Yang, L.; Zhou, Y.-K.; Wang, D.; Li, W.-X. High-Speed Signal Optimization at Differential VIAs in Multilayer Printed Circuit Boards. *Electronics* **2024**, *13*, 3377. [CrossRef]
7. Ji, Q.; Wang, Y.; Xie, M.; Cui, J. Research on fault-tolerance of analog circuits based on evolvable hardware. In *International Conference on Evolvable Systems*; Springer: Berlin/Heidelberg, Germany; pp. 100–108.
8. Deb, K. Multi-objective optimisation using evolutionary algorithms: An introduction. In *Multi-Objective Evolutionary Optimisation for Product Design and Manufacturing*; Springer: London, UK, 2011; pp. 3–34.
9. Azzouz, R.; Bechikh, S.; Said, L.B. Dynamic multi-objective optimization using evolutionary algorithms: A survey. In *Recent Advances in Evolutionary Multi-Objective Optimization*; Springer International Publishing: Cham, Switzerland, 2016; pp. 31–70.
10. Bessaou, M.; Siarry, P. A genetic algorithm with real-value coding to optimize multimodal continuous functions. *Struct. Multidiscip. Optim.* **2001**, *23*, 63–74. [CrossRef]
11. Treadgold, N.K.; Gedeon, T.D. The SARPROP algorithm, a simulated annealing enhancement to resilient back propagation. In Proceedings of the International Panel Conference on Soft and Intelligent Computing, Budapest, Hungary, 7–10 October 1996; pp. 293–298.
12. Bernal, J.; Torres-Jimenez, J. SAGRAD: A program for neural network training with simulated annealing and the conjugate gradient method. *J. Res. Natl. Inst. Stand. Technol.* **2015**, *120*, 113. [CrossRef] [PubMed]
13. Haddow, P.C.; Tyrrell, A.M. Challenges of evolvable hardware: Past, present and the path to a promising future. *Genet Program Evolvable Mach.* **2011**, *12*, 183–215. [CrossRef]
14. Langeheine, J. Intrinsic Hardware Evolution on the Transistor Level. Doctoral Dissertation, University Heidelberg, Heidelberg, Germany, 2005; p. HD-KIP 05-12.
15. Weihmann, T. Geneserver. Master's Thesis, University of Bremen, Bremen, Germany, 2007.
16. Campilho-Gomes, M.; Tavares, R.; Goes, J. Analog flat-level circuit synthesis with genetic algorithms. *IEEE Access* **2024**, *12*, 115532–115545. [CrossRef]
17. Angelis, D.; Sofos, F.; Karakasidis, T.E. Artificial intelligence in physical sciences: Symbolic regression trends and perspectives. *Arch. Comput. Methods Eng.* **2023**, *30*, 3845–3865. [CrossRef] [PubMed]
18. Johar, F.M.; Azmin, F.A.; Suaidi, M.K.; Shibghatullah, A.S.; Ahmad, B.H.; Salleh, S.N. A review of genetic algorithms and parallel genetic algorithms on graphics processing unit (GPU). In *2013 IEEE International Conference on Control System, Computing and Engineering*; IEEE: New York, NY, USA, 2013.
19. Cheng, J.R.; Gen, M. Accelerating genetic algorithms with GPU computing: A selective overview. *Comput. Ind. Eng.* **2019**, *128*, 514–525. [CrossRef]
20. Bosse, S. Design of Analog Computers: Building Blocks and Advanced Methods for Tiny Analog Machine Learning with Surrogate Modeling. In *Proceedings of the 7th International Conference on System-Integrated Intelligence*; Elsevier: Amsterdam, The Netherlands, 2026; pp. 223–237.
21. Sideris, P.; Siskos, S.; Malliaras, G. Verilog-A Modeling of Organic Electrochemical Transistors. In Proceedings of the 2017 6th International Conference on Modern Circuits and Systems Technologies (MOCAST), Thessaloniki, Greece, 4–6 May 2017.
22. Bornemann, S.; Polle, C.; Bosse, S.; May, D.; Skowrons, M.; Lüssum, B. Integration of Organic-Electrochemical Transistors into Composite Material for Structural Health Monitoring. In *Proceedings of the 7th International Conference on System-Integrated Intelligence*; Procedia Computer Science; Elsevier: Amsterdam, The Netherlands; Volume 2026, pp. 126–137.
23. Goyal, K. Operational Amplifier Based Real Time Drift Compensation Circuit for Smart Sensors. In Proceedings of the 14th International Workshop on Computer Science and Engineering (WCSE 2024), Phuket Island, Thailand, 19–21 June 2024; pp. 243–250.
24. Nagel, L.W.; Pederson, D.O. *SPICE (Simulation Program with Integrated Circuit Emphasis)*; Technical Report; UCB/ERL M382 (April 1973); University of California: Berkeley, CA, USA, 1973.
25. Ngspice, an Open Source Spice Simulator for Electric and Electronic Circuits. Available online: https://ngspice.sourceforge.io (accessed on 1 July 2026).
26. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. [CrossRef]
27. Nocedal, J.; Wright, S.J. *Numerical Optimization*; Springer: Berlin/Heidelberg, Germany, 2006.
28. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
29. Duchi, J.; Hazan, E.; Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.* **2011**, *12*, 2121–2215.
30. Kirkpatrick, S.; Gelatt, D., Jr.; Vecchi, M.P. Optimization by simulated annealing. *Science* **1983**, *220*, 671–680. [CrossRef] [PubMed]
31. Metropolis, N.; Rosenbluth, A.W.; Rosenbluth, M.N.; Teller, A.H.; Teller, E. Equation of state calculations by fast computing machines. *J. Chem. Phys.* **1953**, *21*, 1087–1092. [CrossRef]

32. Sampson, J.R. *Adaptation in Natural and Artificial Systems*; Holland, J.H., Ed.; The MIT Press: Cambridge, MA, USA, 1976.
33. Miller, B.L.; Goldberg, D.E. Genetic algorithms, tournament selection, and the effects of noise. *Complex Syst.* **1995**, *9*, 193–212.
34. Kennedy, J.; Eberhart, R. Particle swarm optimization. In *Proceedings of ICNN'95—International Conference on Neural Networks*; IEEE: New York, NY, USA, 1995; Volume 4, pp. 1942–1948.
35. Shi, Y.; Eberhart, R. A modified particle swarm optimizer. In Proceedings of the Evolutionary Computation Proceedings, Anchorage, AK, USA, 4–9 May 1998; Volume 890, pp. 69–73.
36. Kennedy, J.; Mendes, R. Population structure and particle swarm performance. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*; IEEE: New York, NY, USA, 2002; Volume 2, pp. 1671–1676.
37. Maadi, M. Evaluation of SPICE P-N Junction's, BJT's and MOSFET's Model Parameters. In Proceedings of the International Scientific Conference of Iranian Academics, Ankara, Turkey, 5–7 October 2013.
38. Petrosyants, K.O.; Sambursky, L.M.; Kozhukhov, M.V.; Ismail-Zade, M.R.; Kharitonov, I.A.; Li, B. SPICE compact BJT, MOSFET, and JFET models for ICs simulation in the wide temperature range (from −200 °C to +300 °C). *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2020**, *40*, 708–722. [CrossRef]
39. Friedleina, J.T.; McLeoda, R.R.; Rivnay, J. Device physics of organic electrochemical transistors. *Org. Electron.* **2018**, *63*, 398–414. [CrossRef]
40. Kempen, J.V. Design of Inkjet Printed Transistor Devices with Adapated Printing Technology. Bachelor's Thesis, University of Bremen, Bremen, Germany, 2015.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.