

Research Article

A Chaotic Multi-Objective Runge–Kutta Optimization Algorithm for Optimized Circuit Design

Owen M. Nyandieka  and Davies R. Segera 

Department of Electrical and Information Engineering, University of Nairobi, Nairobi 30197, Kenya

Correspondence should be addressed to Owen M. Nyandieka; owennyandieka@gmail.com

Received 26 June 2023; Revised 1 September 2023; Accepted 30 October 2023; Published 7 December 2023

Academic Editor: Erik Cuevas

Copyright © 2023 Owen M. Nyandieka and Davies R. Segera. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Circuit design plays a pivotal role in engineering, ensuring the creation of efficient, reliable, and cost-effective electronic devices. The complexity of modern circuit design problems has led to the exploration of multi-objective optimization techniques for circuit design optimization, as traditional optimization tools fall short in handling such problems. While metaheuristic algorithms, especially genetic algorithms, have demonstrated promise, their susceptibility to premature convergence poses challenges. This paper proposes a pioneering approach, the chaotic multi-objective Runge–Kutta algorithm (CMRUN), for circuit design optimization, building upon the Runge–Kutta optimization algorithm. By infusing chaos into the core RUN structure, a refined balance between exploration and exploitation is obtained, critical for addressing complex optimization landscapes, enabling the algorithm to navigate nonlinear and nonconvex optimization challenges effectively. This approach is extended to accommodate multiple objectives, ultimately generating Pareto Fronts for the multiple circuit design goals. The performance of CMRUN is rigorously evaluated against 11 multiobjective algorithms, encompassing 15 benchmark test functions and practical circuit design scenarios. The findings of this study underscore the efficiency and real-world applicability of CMRUN, offering valuable insights for tailoring optimization algorithms to the real-world circuit design challenges.

1. Introduction

Designing analog circuits can be frustrating due to the many constraints to be attained for a circuit to function correctly. An engineer aims to develop a circuit that meets certain specifications and the International Electro-technical Commission (IEC) standards [1]. For most integrated circuits, the design of the analog part of the overall circuit takes the more significant portion of the overall design time. This is because analog circuit design involves tuning the circuit parameters through trial and error until the required output is met. This process is time-consuming and may not produce the desired results.

Analog circuit design optimization refers to the improvement of the performance of an analog circuit by adjusting its parameters or components to meet specific performance goals, such as increased accuracy, stability, power efficiency, or reduced noise [2]. This process involves using mathematical models, simulation tools, and optimization algorithms to

identify the best combination of circuit components and parameters to achieve the desired performance characteristics. Optimization may be iterative, with multiple design iterations being evaluated until the desired performance criteria are met [2].

Circuit design requires the engineer to know the components used in the design, and it generally involves three steps: topology selection, component sizing, and layout generation [1]. During topology selection, power loss, power gain, current, temperature, circuit stability, and noise are specified, and constraints are set. Component sizing ensures the circuit design is not bulky and uses the fewest number of components possible to avoid redundancy [1]. Many circuit simulation software, such as Autodesk Eagle, NI Multisim, and LTSpice, allow testing of the circuit parameters to see if the desired results are obtained. This is the most widely used method that involves iterative trial-and-error until the parameters fit. Since the above process is time-consuming, there have been several efforts to develop optimization tools

to speed up analog circuit design [1]. However, traditional optimization tools use direct or gradient search methods that utilize initial guesses. These tools struggle with discrete variables and nonlinear constraints and get stuck in local minima when optimizing complex, nonlinear, or nonconvex circuit design problems [1].

Metaheuristic optimization algorithms (MOAs), divided into three classes, namely physics-based, evolutionary, and swarm-based algorithms, replaced the traditional optimization tools for circuit design optimization, especially in handling complex design problems, saving time and resources. Metaheuristics are simply guidelines used to establish rules when solving optimization problems. These rules are referred to as heuristics and are the basic foundations of algorithms. Despite their high performance in producing optimal solutions, MOAs experience several limitations, such as increased sensitivity and difficulty setting control parameters.

The main problem for all MOAs is balancing the exploration and exploitation phases for optimum performance [3]. The exploration phase involves obtaining the global optimal solutions, while the exploitation phase is when the algorithm searches for the local optimal solutions. Algorithms stuck in local optima experience premature convergence, leading to inaccurate results [4]. Convergence is the rate at which an algorithm converges at the global optimum solution. MOAs are stochastic since they use randomly generated components in their optimization process; hence, achieving the appropriate balance between the two phases is difficult [3]. Most researchers try to improve this balance by combining MOAs with other optimizers or choosing control parameters appropriately [4].

In 2021, Ahmadianfar et al. [3] proposed the Runge–Kutta optimization algorithm (RUN) as a more robust alternative to metaphor-based optimization algorithms. The RUN maintains a better balance between the exploration and exploitation phases [3]. This algorithm is divided into two parts. The first part is the search mechanism that utilizes slope calculations of the Runge–Kutta (RK) method to provide a more powerful search space for optimization, and the second part is the enhanced solution quality (ESQ), which improves the algorithm's efficiency by producing more quality solutions than the initially obtained solutions [3].

Despite the RUN showing superior performance compared to metaphor-based algorithms, it still suffers from premature convergence for some problems; thus, Yıldız et al. [5] proposed using chaos to enhance the base RUN, making it chaotic (CRUN) and improving its performance by increasing the convergence rate while maintaining the balance between exploration and exploitation phases of the algorithm. The CRUN was applied to the real-world design problems and showed superior performance regarding the diversity of solutions and convergence speed [5]. However, both the RUN and CRUN algorithms are single-objective, and circuit design involves optimizing multiple objectives, such as increasing power efficiency while minimizing cost.

This paper proposes a chaotic multi-objective Runge–Kutta optimization algorithm (CMRUN) for circuit design optimization. Introducing chaos into the base RUN enhances its

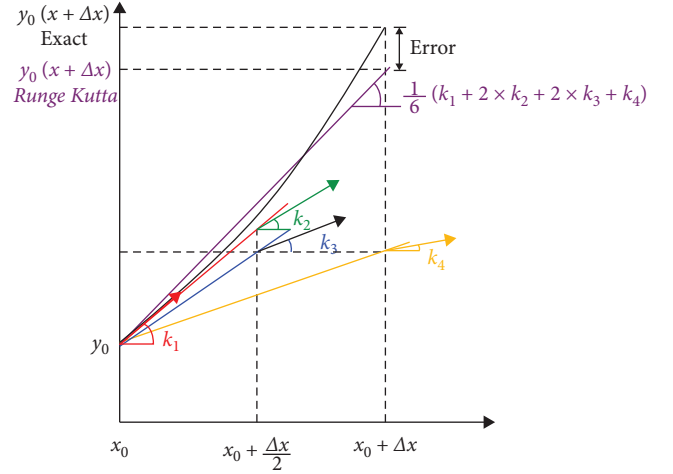


FIGURE 1: Slopes utilized in the Runge–Kutta Method.

performance by improving the balance between the exploitation and exploration phases and, thus, a high-convergence rate while avoiding local optima. Furthermore, multi-objective optimization techniques are incorporated into this chaotic RUN, allowing the optimization of multiple circuit performance metrics simultaneously [6].

The organization of this paper is as follows: Section 2 explains the formulation and optimization methods of the RUN algorithm, its limitations and methods undertaken to improve it. Section 3 gives the background of the abstract ideas (chaos, multi-objective optimization, and circuit design) used to complete this work. Section 4 describes the methods used to design the CMRUN, the performance measurement criteria, and the circuit parameters to be optimized. Section 5 evaluates the performance of the CMRUN in optimizing benchmark functions and circuit parameters. Section 6 provides the conclusions about this work and suggestions for the future research.

2. Literature Review

2.1. The Runge–Kutta Optimization Algorithm (RUN)

2.1.1. Inspiration behind the Runge–Kutta Optimization Algorithm. The RK method is one of the techniques in numerical methods used to find solutions for first-order ordinary differential equations:

$$\frac{dy}{dt} = f(x, y), y(x_0) = y_0. \quad (1)$$

The slope of the line of best fit at point (x, y) is defined as $f(x, y)$ in the equation above, giving the main idea of what the algorithm will base its search space on. Figure 1 shows the slopes utilized in the RK method.

As shown below, the RK method can be derived using the Taylor series and ignoring the higher order terms.

$$y(x + \Delta x) = y(x) + y'(x)\Delta x + y''(x)\frac{(\Delta x)^2}{2!} + \dots \quad (2)$$

The equation below is the approximate solution for the Taylor series derivation of the RK method after dropping the higher order terms:

$$y(x + \Delta x) \approx y(x) + y'(x)\Delta x. \quad (3)$$

From Equation (3), we can rewrite the equation as shown below to begin finding the formula for the RK fourth-order method:

$$y(x + \Delta x) = y(x) + k_1\Delta x, \quad (4)$$

where $k_1 = y'(x) = f(x, y)$; and $\Delta x = x_{n+1} - x_n$.

The first-order derivative, which is equivalent to k_1 as seen above, can be determined as shown in the expression below:

$$y'(x) = \frac{y(x + \Delta x) - y(x - \Delta x)}{2}. \quad (5)$$

Therefore, we can use Equation (5) to rewrite Equation (4) as shown below:

$$y(x + \Delta x) = y(x) + \frac{y(x + \Delta x) - y(x - \Delta x)}{2}. \quad (6)$$

The solutions using this method are formulated as shown below:

$$y(x + \Delta x) = y(x) + \frac{1}{6}(k_1 + 2 \times k_2 + 2 \times k_3 + k_4)\Delta x, \quad (7)$$

where

$$\begin{aligned} k_1 &= y'(x) = f(x, y) \\ k_2 &= f\left(x + \frac{\Delta x}{2}, y + \frac{\Delta x}{2} \times k_1\right) \\ k_3 &= f\left(x + \frac{\Delta x}{2}, y + \frac{\Delta x}{2} \times k_2\right) \\ k_4 &= f(x + \Delta x, y + \Delta x \times k_3). \end{aligned} \quad (8)$$

From the equations above, k_1 uses y to give the slope at the beginning of the interval $[x, x + \Delta x]$. k_2 is defined by the midpoint slope using y and k_1 [3]. k_3 is defined by the midpoint slope using y and k_2 , and k_4 is determined by the pitch at the end using y and k_3 . Remember, from the Runge–Kutta method, the value $y(x + \Delta x)$ is given by the initial value $y(x)$ and the weighted averages of k_1, k_2, k_3 , and k_4 . This is depicted in Figure 2. Figures 1 and 2 are obtained from the study of Ahmadianfar et al. [3].

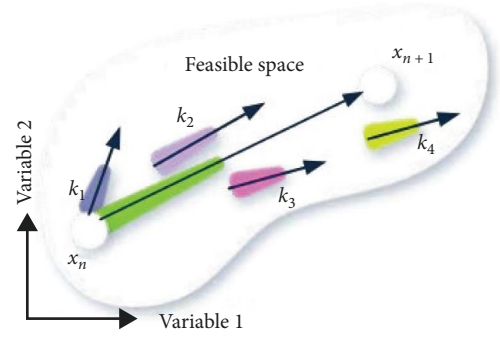


FIGURE 2: Slopes used to determine x_{n+1} in the RUN algorithm.

2.1.2. Optimization Steps of the RUN. This algorithm uses random components in a swarm-based model, which is population-based, and the RK logic to develop ordinary differential equations used to find the slope [3]. This slope is used as the search mechanism to explore and find the best solutions while observing the rules of the evolution of the swarm-based model. The RK optimizer is mathematically formulated in the sections below.

(1) *Initialization of the RUN Algorithm.* An initial swarm will undergo evolution for several iterations to start the algorithm. Therefore, initializing a size N population means N random positions are generated [3, 5]. The solutions of the optimization problem are the members of the population x_n ($n = 1, 2, \dots, N$) with a dimension D . The idea below is used in generating the random initial solutions in the RUN algorithm.

$$x_{n,l} = L_l + \text{rand} \cdot (U_l - L_l). \quad (9)$$

L_l —the lower limit of the l -th parameter and U_l —upper limit of the l -th parameter where $l = (1, 2, \dots, D)$ rand—random number in $[0, 1]$.

(2) *Development of a Searching Mechanism Using the Runge–Kutta Method.* The RUN algorithm's searching mechanism is centered on the RK method and uses random solutions to perform searches in the search space, both globally and locally [3, 5, 7]. Like every other optimizer, this is the core of the program. From Equation (5) of the RK method, the neighbors of x_n are:

$x_n - \Delta x$ —best position and $x_n + \Delta x$ —worst position.

We can use Equation (5) to come up with the first coefficient, k_1 , defining it as follows:

$$k_1 = \frac{x_w - x_b}{2\Delta x}. \quad (10)$$

x_w —worst solution at each iteration; x_b —best solution at each iteration; x_w and x_b are determined by selecting three random solutions from the population. These three solutions are: (x_{r1}, x_{r2}, x_{r3}) and $r1 \neq r2 \neq r3 \neq n$.

From the equation of k_1 , we can see that it lacks stochastic behavior. Therefore, to improve the exploration search

phase, we introduce randomness and rewrite the equation as follows:

$$k_1 = \frac{1}{2\Delta x}(\text{rand} \times x_w - u \times x_b), \quad (11)$$

rand—random number in the range [0,1]

The best solution (x_b) is essential in enhancing the global search to find the global best solution. Thus, parameter u is introduced to grow the best solution's importance [3].

$$u = \text{round}(1 + \text{rand}) \times (1 - \text{rand}). \quad (12)$$

The range between adjacent positions is given by:

$$\Delta x = 2 \times \text{rand} \times |\text{Stp}|. \quad (13)$$

The step size (Stp) is given by the equation:

$$\text{Stp} = \text{rand} \times ((x_b - \text{rand} \times x_{\text{avg}}) + \gamma). \quad (14)$$

γ —is the scale factor determined by:

$$\gamma = \text{rand} \times (x_n - \text{rand} \times (u - l)) \times \exp\left(-4 \times \frac{i}{\text{Maxi}}\right). \quad (15)$$

The scale factor decreases exponentially during optimization and is determined by the solution space size. The randomness of the numbers in Equations (13)–(15) provides diversification in the searches [3].

The rest of the coefficients k_2 , k_3 , and k_4 are written below:

$$k_2 = \frac{1}{2\Delta x}(\text{rand} \cdot (x_w + \text{rand}_1 \cdot k_1 \cdot \Delta x) - (u \cdot x_b + \text{rand}_2 \cdot k_1 \cdot \Delta x)), \quad (16)$$

$$k_3 = \frac{1}{2\Delta x} \left(\text{rand} \cdot \left(x_w + \text{rand}_1 \cdot \left(\frac{1}{2} k_2 \right) \cdot \Delta x \right) - \left(u \cdot x_b + \text{rand}_2 \cdot \left(\frac{1}{2} k_2 \right) \cdot \Delta x \right) \right), \quad (17)$$

$$k_4 = \frac{1}{2\Delta x}(\text{rand} \cdot (x_w + \text{rand}_1 \cdot k_3 \cdot \Delta x) - (u \cdot x_b + \text{rand}_2 \cdot k_3 \cdot \Delta x)). \quad (18)$$

The numbers rand_1 and rand_2 are random numbers in the range [0,1]. The program below determines the worst and best solutions: x_w and x_b .

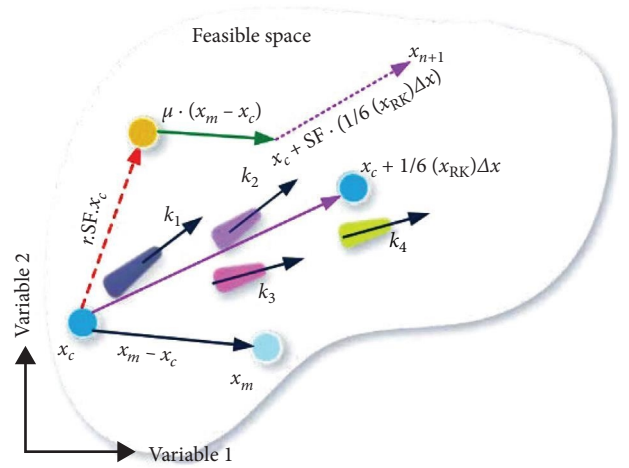


FIGURE 3: The search mechanism of the RUN.

$$\begin{aligned} & \text{if } f(x_n) < f(x_{bi}) \\ & \quad x_b = x_n \\ & \quad x_w = x_{bi} \\ & \text{else} \\ & \quad x_b = x_{bi} \\ & \quad x_w = x_n \\ & \text{end} \end{aligned} \quad (19)$$

x_{bi} is the best random solution from random selections (x_{r1} , x_{r2} , and x_{r3})

The searching mechanism of the RUN is shown in Figure 3, and is given by:

$$\text{SM} = \frac{1}{6} (x_{\text{RK}}) \cdot \Delta x, \quad (20)$$

in which

$$x_{\text{RK}} = k_1 + 2 \times k_2 + 2 \times k_3 + k_4. \quad (21)$$

(3) *Modernizing the Solutions.* During optimization, the RUN algorithm updates the positions of the solutions at each iteration using the RK method since the initial solutions are random selections [3]. The following pseudocode shows how the solutions are updated in the exploitation and exploration phases.

$$\begin{aligned} & \text{if } \text{rand} < 0.5 \\ & \quad (\text{exploration phase}) \\ & \quad x_{n+1} = (x_c) + \text{SF} + \text{SM} + \mu \times x_s \\ & \text{else} \\ & \quad (\text{exploitation phase}) \\ & \quad x_{n+1} = (x_m) + \text{SF} \times \text{SM} + \mu \times x_{s'} \\ & \quad \text{end,} \end{aligned} \quad (22)$$

where

$$\mu = 0.5 + 0.1 \times \text{randn}, \quad (23)$$

where μ is a random number, and randn is a random number with a normal distribution.

The expressions for x_s and $x_{s'}$ are:

$$x_s = \text{randn} \cdot (x_m - x_c), \quad (24)$$

$$x_{s'} = \text{randn} \cdot (x_{r1} - x_{r2}). \quad (25)$$

The expressions for x_c and x_m are as follows:

$$x_c = \varphi \times x_n + (1 - \varphi) \times x_{r1}, \quad (26)$$

$$x_m = \varphi \times x_{\text{best}} + (1 - \varphi) \times x_{\text{ibest}}, \quad (27)$$

where φ is a random number in $[0, 1]$. x_{best} is the current best solution. x_{ibest} is the best solution at each iteration.

$$\text{SF} = 2 \cdot (0.5 - \text{rand}) \times f, \quad (28)$$

$$f = a \times \exp\left(-b \times \text{rand} \times \left(\frac{i}{\text{Maxi}}\right)\right). \quad (29)$$

The adaptive factor, SF, is the one that provides the balance needed between exploitation and exploration [3, 5]. Initially, SF is large to enhance exploration by increasing diversity, and it decreases in value afterward, thus increasing the number of iterations and enhancing exploitation.

a and b are the main control parameters of the SF. b and c are constants; i —number of iterations; Maxi—maximum number of iterations.

From Equation (22), when $\text{rand} < 0.5$ in the solution space, the algorithm conducts a global search, and simultaneously, around solution x_c , it does a local search. The exploration phase ensures all high-quality solutions are searched. When $\text{rand} \geq 0.5$, the RUN applies a local search around x_m . Exploitation increases the speed of convergence, focusing on promising solutions. This is done by rewriting Equation (22) as below, where,

r —integer number (either 1 or -1); g —random number in $[0, 2]$; r increases diversity by changing the search directions, and exploitation around x_c reduces with each iteration.

if $\text{rand} < 0.5$

(exploration phase)

$$x_{n+1} = (x_c + r \cdot \text{SF} \cdot g \cdot x_c) + \text{SF} \cdot \text{SM} + \mu \cdot \text{randn} \cdot (x_m - x_c)$$

else

(exploitation phase)

$$x_{n+1} = (x_m + r \cdot \text{SF} \cdot g \cdot x_m) + \text{SF} \cdot \text{SM} + \mu \cdot \text{randn} \cdot (x_{r1} - x_{r2})$$

end

(30)

(4) *Enhanced Solution Quality*. This ensures solutions after each iteration head toward better positions. It operates using the following pseudocodes:

if $\text{rand} < 0.5$

if $w < 1$

$$x_{\text{new2}} = x_{\text{new1}} + r \cdot w \cdot |(x_{\text{new1}} - x_{\text{avg}}) + \text{randn}|$$

else

$$x_{\text{new2}} = (x_{\text{new1}} - x_{\text{avg}}) + r \cdot w \cdot |(u \cdot x_{\text{new1}} - x_{\text{avg}})' + \text{randn}|$$

end

end

(31)

in which

$$w = \text{rand}(0, 2) \cdot \exp\left(-c \left(\frac{i}{\text{Maxi}}\right)\right), \quad (32)$$

$$x_{\text{avg}} = \frac{x_{r1} + x_{r2} + x_{r3}}{3}, \quad (33)$$

$$x_{\text{new1}} = \beta \times x_{\text{avg}} + (1 - \beta) \times x_{\text{best}}, \quad (34)$$

β —is a random number in $[0, 1]$; c —random number is equal to $5 \times \text{rand}$; w —random number (decreases with increase in iterations); r —integer number (either 1 or 0 or -1).

The solution (x_{new2}) may not be better than the current solution; hence, the following equation gives the algorithm another chance to find a better solution than the two [3].

if $\text{rand} < w$

$$x_{\text{new3}} = (x_{\text{new2}} - \text{rand} \cdot x_{\text{new2}}) + \text{SF} \cdot (\text{rand} \cdot x_{\text{RK}} + (\nu \cdot x_b - x_{\text{new2}}))$$

end,

(35)

where $\nu = 2 \times \text{rand}$.

The implementation of (x_{new3}) occurs when $\text{rand} < w$ to move (x_{new2}) to a better position. The parameter ν enhances the importance of the best solution and to find x_{RK} ; x_b and

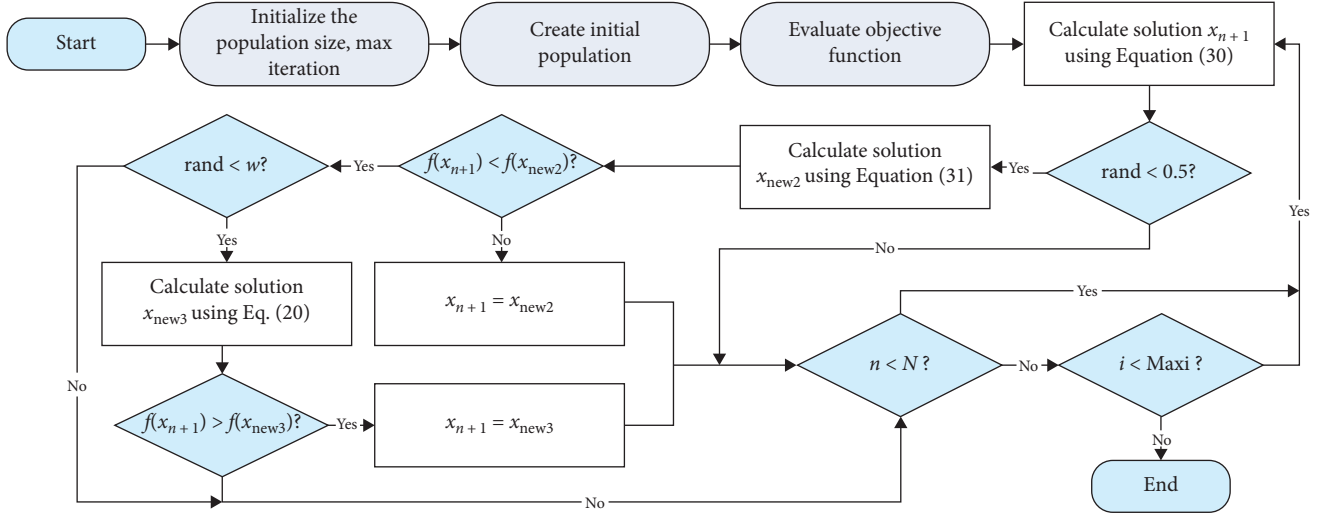


FIGURE 4: Flowchart of the run algorithm.

x_w , respectively, become x_k and x_{new2} since the fitness of x_n is lower than that of x_{new2} . The flowchart of the RUN algorithm is shown in Figure 4 [3].

(5) *The Pseudocode for the RUN Algorithm.*

2.1.3. Limitations of Runge–Kutta Optimizer. Despite the RUN algorithm having better convergence speeds due to appropriate balancing between the local and global searches, it still suffers some limitations similar to the metaphor-based optimizers mentioned below:

- (i) The algorithm is single-objective based and cannot solve multi-objective optimization problems [3].
- (ii) The algorithm can be trapped in the local search during ESQ, confining it to only locally optimal solutions [3].

2.2. Methods Undertaken to Improve the RUN's Performance. The RUN is a fairly new optimizer designed in 2021. This algorithm has demonstrated superior performance to traditional metaphor-based algorithms, and thus, several researchers have proposed ways to improve its performance even more [7]. The RUN is a stochastic population-based algorithm [5]. Unlike single-based optimizers, population-based algorithms randomly generate solutions at each iteration to avoid being trapped in local optima. Therefore, they are superior to single-based algorithms as they attain quality solutions and have increased convergence speed. The randomly generated solutions can share information, ensuring convenient search in complex feature space sceneries [5]. Table 1 shows the pseudocode for the RUN algorithm.

The RUN algorithm edges nature-inspired metaheuristic algorithms in performance. Generally, metaheuristic algorithms are divided into three: evolutionary algorithms (EAs), swarm intelligence algorithms (SIAs), and physics-based algorithms (PBAs) [3]. These algorithms have shown a high capability at obtaining optimum solutions at considerable speed while avoiding getting trapped in local optima. However,

Part 1. Initialization

Initialization *parameters*

Randomly generate the initial population of the RUN

Evaluate the objective function for each population member

Obtain x_w , x_b , and x_{best}

Part 2. RUN operators

for $it = 1 : MaxIt$

for $i = 1 : np$

for $j = 1 : dim$

Evaluate position $x_{n+1,l}$ (Equation 30)

end for

ESQ

if $rand < 0.5$

Evaluate position x_{new2} (Equation 31)

if $f(x_n) < f(x_{new2})$

if $rand <$

Evaluate position x_{new3} (Equation 35)

end

end

end

Modernize positions x_w and x_b

end for

Modernize position x_{best}

$it = it + 1$

end

Part 3. return x_{best}

ALGORITHM 1: The Pseudocode of RUN.

when solving complex design problems, they tend to suffer from premature convergence and mismatch between the design variables. Over the years, researchers developed hybrid optimizers to fix the issues encountered by metaheuristics by combining the domineering features of individual algorithms.

TABLE 1: List of chaotic maps.

Sr. no.	Map	Fundamental mathematical form	Nomenclatures
1	Chebyshev	$p_{k+1} = \cos(k \cos^{-1}(p_k))$	p = Location/opted position in search space k = Iteration
2	Circle	$p_{k+1} = p_k + b = (a - 2\pi)\sin(2\pi p_k) \bmod(1)$	$a = 0.5$ $b = 0.2$
3	Gauss/mouse	$p_{k+1} = \begin{cases} \frac{\varepsilon + p_k + cC_k^n}{1 + R}, & R < p_k < 1 \\ 1 / \frac{1}{p_k \bmod(1)} = \frac{1}{p_k} - \lfloor \frac{1}{p_k} \rfloor \end{cases}$ $0 < p_k \leq R$	R = Range for the position
4	Iterative	$p_{k+1} = \sin(\frac{\lambda \pi}{p_k})$	λ = Parameter ranges from 0 to 1
5	Logistic	$p_{k+1} = \alpha p_k(1 - p_k)$	
6	Piecewise	$f(p) = \begin{cases} \frac{p_k}{R} & 0 \leq p_k < R \\ \frac{p_k - R}{0.5 - R} & R \leq p_k < \frac{1}{2} \\ \frac{1 - R - p_k}{1 - R - \frac{1}{2}} & \frac{1}{2} \leq p_k < 1 - R \\ \frac{0.5 - R}{1 - p_k} & 1 - R \leq p_k < 1 \\ \frac{R}{1 - p_k} & 1 - R \leq p_k < 1 \end{cases}$	
7	Sine	$p_{k+1} = \frac{a}{4} \sin(\pi p_k)$	μ = Parameter count ranges from 0.9 to 1.08 $a = 2.3$ $p_0 = 0.7$
8	Singer	$p_{k+1} = \mu(7.86p_k - 23.31p_k^2 + 28.75p_k^3 - 13.3028.75p_k^4)$	
9	Sinusoidal	$p_{k+1} = \alpha p_k^2 \sin(\pi p_k)$ $p_{k+1} = \sin(\pi p_k)$	
10	Tent	$p_{k+1} = \begin{cases} \frac{p_k}{0.7}, & p_k < 0.7 \\ \frac{10}{3}(1 - p_k), & p_k \geq 0.7 \end{cases}$	

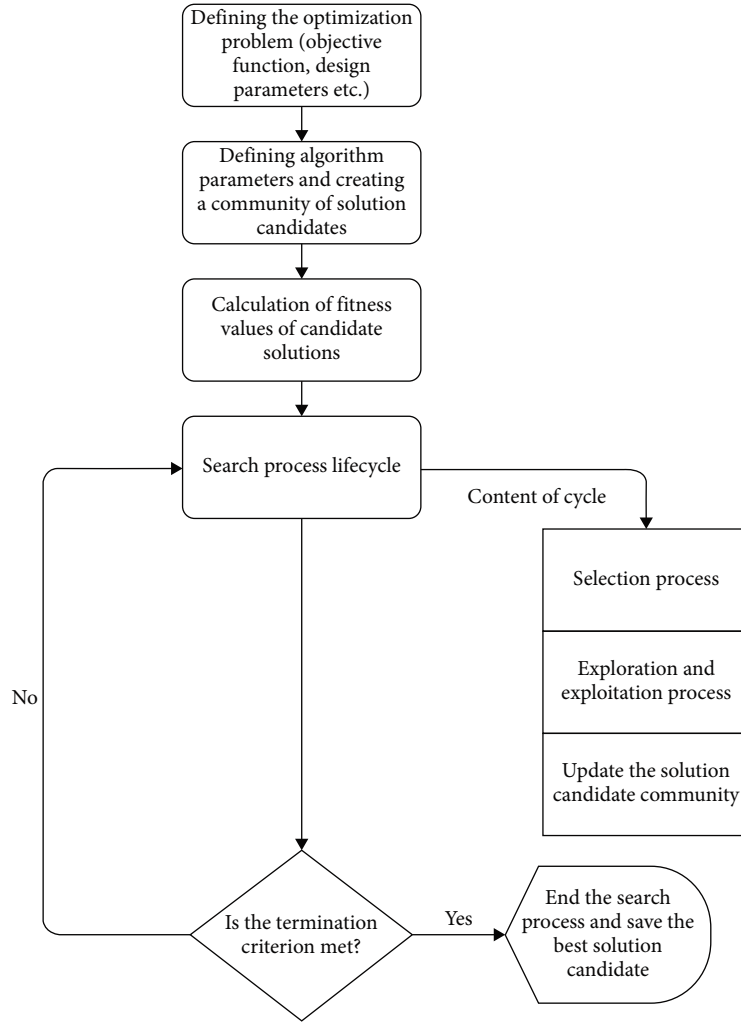


FIGURE 5: General flowchart for metaheuristic algorithm.

An example is the hybrid of the whale optimization algorithm and the graygrey wolf optimizer, which showed better performance than the individual algorithms [8]. Figure 5 shows the general flowchart of metaheuristic algorithms [9].

The RUN algorithm was designed to be a powerful and more accurate optimizer that avoids local optima. The RUN finds the global best in solving optimization problems using the ESQ to increase the convergence speed and avoid premature convergence [3]. Although the RUN performs better than other metaheuristic algorithms, it suffers limitations when solving multimodal problems [3]. To solve this problem, researchers have proposed several methods of improving the RUN's performance. Some of these methods are discussed below.

CENGİZ et al. [9] proposed a method to improve the RUN algorithm using the FDB (fitness distance balance) method. In the RUN algorithm, local minima traps can occur during execution; hence, the global optimal solutions are not attained. The FDB method was proposed to enhance the exploration phase, guiding the algorithm toward global solutions. There are 10 cases presented by CENGİZ et al. [9] to modify the base RUN by incorporating FDB to enhance its

performance. Some of these cases include modifying Equation (33) in the following ways:

$$x_{\text{avg}} = \frac{x(\text{fdb}, :) + x(\text{fdb}, :) + x(\text{fdb}, :)}{3}, \quad (36)$$

or

$$x_{\text{avg}} = \frac{x(\text{fdb}, :) + x(\text{fdb}, :) + x(C, :)}{3}, \quad (37)$$

where fdb is the FDB operator.

The FDBRUN produced better results compared to the base algorithm in the study conducted by CENGİZ et al. [9]. The readers are encouraged to read this paper [9] to view the other cases and how they are implemented.

Devi et al. [10] proposed another way to improve the RUN algorithm to avoid premature convergence and enhance the accuracy of solutions. They proposed integrating a local escaping operator (LEO) in the RUN [10]. The LEO improves it by bypassing local minima and thus increasing its convergence. The readers are encouraged to read this paper by Devi et al.

[10] to understand better the formulation of the LEO and its integration in the base RUN.

The above methods improve the performance of the RUN algorithm, but the most widely used method to improve metaheuristic algorithms is enhancing them using chaotic maps. Several optimizers have been improved by introducing chaos, such as the chaotic Mayfly algorithm, the chaotic whale optimizer, and the chaotic bat algorithm [11–15]. These metaheuristic algorithms' performance and quality parameters have been tested by applying them to real-world design problems. For example, the chaotic Harris Hawks optimizer developed by Gezici and Livatyali [12] provided better performance and results than the traditional Harris Hawks optimizer [16].

Therefore, from existing research, Yıldız et al. [5] proposed using chaos to enhance the base RUN by making it chaotic (CRUN) to increase its ability to avoid local optima. The CRUN was applied to the real-world design problems and showed superior performance regarding the diversity of solutions and convergence speed compared to the original RUN [5].

In this paper, the approach used to design the CMRUN is similar to that used by Yıldız et al. [5] to design the CRUN. Chaotic maps are integrated into the core RUN structure to strike a refined balance between exploration and exploitation, enabling the algorithm to navigate nonconvex and complex optimization problems. Furthermore, utilizing chaos in the ESQ improves the convergence speed while navigating the algorithm toward global solutions [5]. The algorithm is then modified to generate Pareto Fronts to handle complex problems with multiple objectives [6].

3. Theoretical Background

3.1. Chaotic Maps. In essence, chaotic maps are mathematical functions that behave chaotically, such that slight variations in the original conditions have radically diverse effects. Chaotic maps are typically utilized in control and optimization algorithms to broaden the range of potential solutions and avoid local minima traps [11]. The Logistic, Tent, Circular, Gaussian, and Chebyshev maps are examples of chaotic maps.

Due to their tendency to get stuck in local optima, traditional algorithms struggle to optimize nonconvex and multimodal objective functions. By adding unpredictability to the optimization process, chaotic maps offer a solution to this issue [12–15]. The optimization algorithm can better traverse the search space and avoid local optima by using chaotic maps to produce random solutions or perturb the present solution. This is especially helpful in high-dimensional search spaces, where many local optima may exist [5].

Chaotic maps can be incorporated in several ways into optimization algorithms. One method is to use chaotic maps to produce the initial population of solutions in an evolutionary algorithm. A group of random solutions can be produced using the chaotic map, and these solutions can then be evolved by crossover, mutation, and selection processes [15].

Another approach is when a local search algorithm like simulated annealing or tabu search uses chaotic maps to alter

the existing solution. The chaotic map can produce a minor perturbation to the present solution, which is accepted or rejected according to a probability distribution [11].

Finally, the parameters of a chaotic system itself can also be optimized using chaotic maps. The chaotic map changes the system's parameters and alters the behavior of the objective function as it determines the system's complexity or randomness [5].

Overall, chaotic maps provide an effective tool for exploring complex, nonlinear search spaces and avoiding local optima in the optimization algorithms. Chaotic maps can enhance the effectiveness and efficiency of optimization algorithms in various applications by bringing randomness and diversity into the optimization process [16]. Randomization helps to balance the exploitation and exploration phases for optimal performance [15].

In this paper, 10 chaotic maps are integrated to determine the best chaotic map in circuit design. They are obtained from [5] and are shown in Table 1.

Figure 6 shows each map's characteristics when implemented in MATLAB.

3.2. Multi-Objective Optimization Techniques. In the engineering world, engineers often encounter several problems with multiple objectives that must be optimized simultaneously. In most cases, these objectives have some sort of conflict, such as improving one objective deteriorates another. Additionally, these objectives have different units of measurement and are called multi-objective optimization problems (MOPs). Unlike in single-objective problems where a single optimal result is obtained by determining which solution is better, MOPs do not have a clear-cut method of determining which solution is better due to conflicts and different measurement units [6].

Circuit design often involves the optimization of multiple objectives, such as the gain output of a system and the cost of components. Thus, in this paper, the RUN algorithm will be made multi-objective to handle several objectives simultaneously. There are several ways of making an algorithm multi-objective for solving MOPs. These methods include many methods as described below.

3.2.1. Weighted Sum Method. This technique involves the combination of several objectives into one objective in MOPs. Each objective is assigned a weight, and then a linear combination is applied to all weighted objectives [6]. The weights reflect the relative importance of each objective and hence can be used to balance the tradeoffs between objectives. For example, consider a MOP with two objectives, $y_1(x)$ and $y_2(x)$. The objectives can be combined using the weighted sum method as shown below:

$$z(x) = w_1 \times y_1(x) + w_2 \times y_2(x), \quad (38)$$

where w_1 and w_2 are the weights of $y_1(x)$ and $y_2(x)$, respectively.

The optimal solution can be found by minimizing $z(x)$ and will depend on the values of w_1 and w_2 , which can be tuned to give the relative importance of each objective.

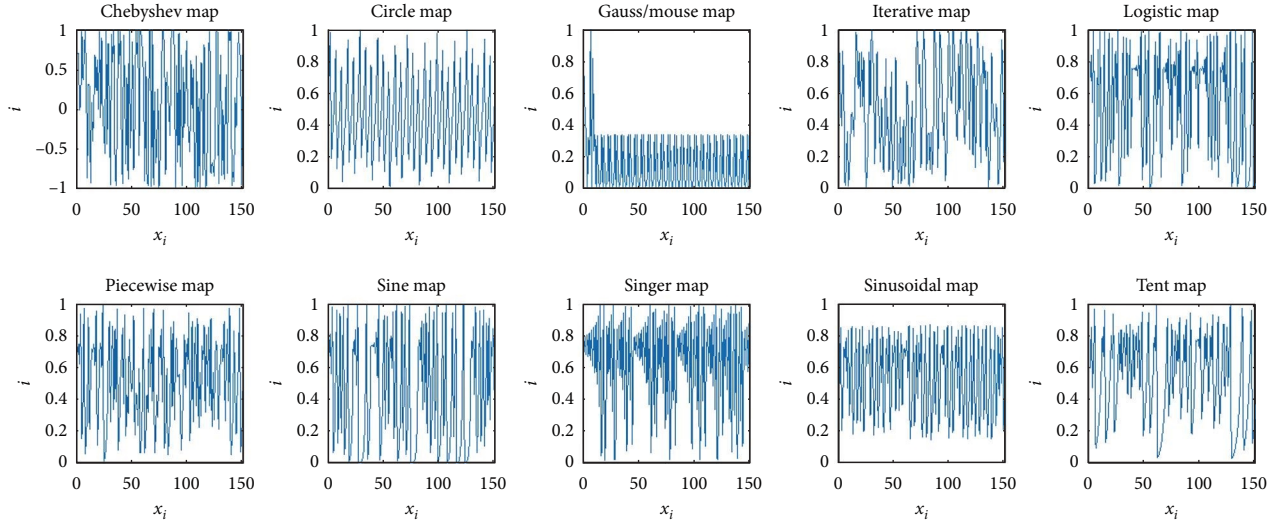


FIGURE 6: MATLAB implementation of chaotic maps.

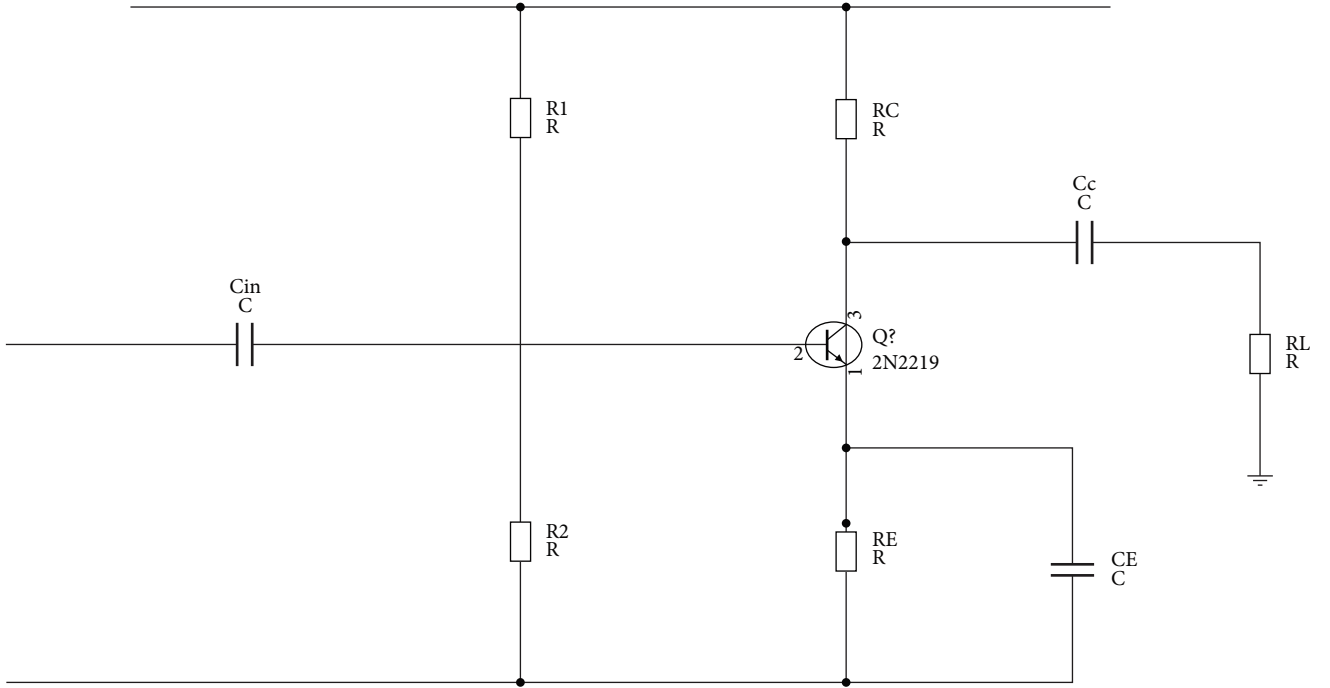


FIGURE 7: Typical circuit for a single-stage amplifier.

3.2.2. Scalarizing Method. This technique involves converting an MOP into a single-objective optimization problem by defining a scalarizing function that maps several objectives into one objective. For example, consider a MOP with two objectives, $y_1(x)$ and $y_2(x)$. The problem is modified to contain one objective, as shown below:

$$f(x) = g(y_1(x), y_2(x)), \quad (39)$$

where $f(x)$ is the scalarizing function.

Based on the specified needs of the problem, the scalarizing function is defined in several ways. For example, it could be a function that reflects the tradeoffs between objectives or as a weighted sum of objectives [6]. The function is then mapped back to the original MOP once an optimal solution is found.

3.2.3. Pareto Optimization. This technique involves finding a set of nondominated solutions, which means that improving one objective has the effect of deteriorating one or more of the other objectives. If no solution exists that is better in at

least one or all objectives, the solution obtained is considered Pareto optimal [17, 18]. These nondominated solutions form the Pareto front, which is a representation of all Pareto optimal solutions.

3.2.4. Evolutionary Algorithms. Natural evolution processes inspire these algorithms and come in handy in solving complex MOPs that have conflicting objectives. The operation of evolutionary algorithms involves the iterative generation of populations of solutions, and the fitness function searches for the optimal solution by evaluating the quality of each solution [6]. Modifying the solutions selected occurs through crossover and mutation, creating a fresh set of solutions. The processes are repeated until the optimal solution is attained or a set condition is achieved [17]. Evolutionary algorithms best solve problems with the wide search spaces and complex nonlinear objectives.

3.2.5. Multi-Objective Gradient Descent Methods. This technique optimizes multiple objectives simultaneously using modified gradient descent algorithms. Here, a multi-objective loss function is defined, which combines all the objectives into a single objective. The loss function reflects the tradeoffs between objectives and should be differentiable to make the gradient computation possible. The multi-objective gradient descent algorithm then adjusts the model parameters to minimize the loss function iteratively, whereby at each iteration, the gradient of the loss function is computed. Then, the model parameters are updated in a way that minimizes loss. This process is repeated until a preset condition is achieved or the loss function converges to a minimum.

3.2.6. Decomposition-Based Methods. These are algorithms that divide MOPs into a series of subproblems and optimize each subproblem separately. After obtaining individual solutions to each subproblem, these solutions are combined into one final solution to the original MOP.

The choice of a multi-objective optimization technique typically depends on the unique aspects of the problem and the optimization's objectives. The choice of technique can significantly affect the effectiveness and quality of the solution since each technique has advantages and disadvantages.

3.3. Optimized Circuit Design. Circuit design is designing electronic circuits by choosing parameters that meet specific performance criteria based on the theoretical expectations. The performance of a circuit is measured using criteria such as noise, stability, speed, and power consumption. Optimized circuit design aims to achieve the desired performance while minimizing the circuit's cost, size, and complexity [1].

There are several approaches to optimized circuit design, each with advantages and limitations. Here are some of the most commonly used techniques:

- (i) Top-down design: this approach involves starting with the overall system requirements and working downwards to the circuit level. The design is broken down into a hierarchy of subcircuits, each designed to meet specific requirements. The advantage of this approach is that it ensures that the design meets the

overall system requirements, but it can be time-consuming and may not always result in the most efficient circuit.

- (ii) Bottom-up design: this approach involves starting with individual circuit components and building the overall system. The advantage of this approach is that it can be faster and more efficient than top-down design, but it may not always result in the best overall performance.
- (iii) Simulation-based design: this approach uses computer simulations to model the circuit's behavior and optimize its performance. Simulation-based design can be very effective at identifying design flaws and optimizing circuit performance, but it can be time-consuming and computationally expensive.
- (iv) Optimization algorithms: optimization algorithms such as genetic algorithms can optimize circuit designs by generating a population of candidate solutions and evaluating their performance using simulations [2].

Of all the methods, optimization algorithms help to save time by giving the optimal solution based on the range of input parameters, eliminating the need for trial-and-error, which is time-consuming.

The circuit design proposed in this paper is a single-stage amplifier. Amplifiers are used to improve the strength of weak signals. Multistage amplifiers provide superior performance by allowing more flexibility for the input and output impedances. Understanding the workings of a single-stage amplifier is crucial since they are connected in a cascade to form multistage amplifiers to provide even higher gain [19].

A single-stage amplifier comprises a single transistor with a bias circuit and other components to facilitate the desired gain output for quantities such as current, voltage, and power [19]. The following components and parts are included for a typical single-stage amplifier design.

- (i) Transistor: the transistor is the active device in the amplifier circuit and is responsible for amplifying the input signal. Different types of transistors can be used depending on the specific requirements of the circuit.
- (ii) Biasing circuit: the biasing circuit sets the operating point of the transistor, which is important for ensuring that the amplifier operates in the linear region and avoids distortion [19]. The biasing circuit typically consists of resistors and capacitors connected to the transistor.
- (iii) Input coupling capacitor: used to isolate the input signal from the DC bias of the amplifier circuit.
- (iv) Output coupling capacitor: the output coupling capacitor isolates the amplifier circuit from the load or the next stage in the circuit by blocking DC signals [19].
- (v) Load resistor: converts the output current into an output voltage. It determines the gain of the

amplifier and is typically chosen to match the load's impedance.

- (vi) Feedback network: the feedback network provides negative feedback to the amplifier, which helps reduce distortion and improve stability. The feedback network typically consists of resistors and capacitors connected between the amplifier's output and input.
- (vii) Power supply: it powers the amplifier circuit. It typically consists of a DC voltage source and filtering capacitors to remove any AC noise or ripple from the supply voltage.

Several parameters can be optimized for optimum performance in designing a single-stage amplifier. These are:

- (i) Gain: increasing the gain of the amplifier can improve its sensitivity and ability to amplify weak signals.
- (ii) Bandwidth: increasing the amplifier's bandwidth can improve its ability to amplify high-frequency signals.
- (iii) Input and output impedance: optimizing the input and output impedance can improve the matching between the amplifier and the signal source and load, respectively.
- (iv) Distortion: distortion is any unwanted modification of the input signal, and it can be caused by nonlinearity in the amplifier circuit. Minimizing distortion can improve the fidelity and accuracy of the output signal.
- (v) Noise: noise is any unwanted signal that the amplifier circuit adds to the output signal. Minimizing noise can improve the output signal-to-noise ratio (SNR).
- (vi) Power consumption: minimizing power consumption can improve the amplifier's efficiency and prolong the battery life in the portable devices.
- (vii) Stability: stability refers to the ability of the amplifier to operate reliably and predictably over time without oscillations or instability. An unstable amplifier can produce unwanted oscillations, which can cause distortion, noise, and other performance issues.

Figure 7 shows a typical single-stage amplifier circuit. Single-stage amplifiers are used in tape recorders, radio, television receivers, CD players, and public address systems. Therefore, the design of optimal single-stage amplifiers is essential in electronics.

4. Methodology

4.1. Chaotic Multi-Objective Runge–Kutta Optimization Algorithm (CMRUN). The following steps were undertaken to improve the base RUN algorithm to handle multiple objectives while incorporating chaos to improve the balance between exploitation and exploration searches.

4.1.1. Selecting What Part of the Base RUN to Improve by Chaos. Many optimization algorithms fall short due to the local minima traps, especially when solving nonconvex and

multimodal problems. Researchers, therefore, came up with a solution to enhance their performance by increasing the diversity of solutions and thus avoiding local minima traps [12–15]. As mentioned in Section 3, several ways to implement chaos in optimization algorithms exist. One way is to use chaotic maps to generate the initial population of solutions, which undergoes selection, crossover, and mutation operations.

Another way is the chaotic map can be used to generate a small perturbation to the current solution, which is then accepted or rejected based on a probability distribution. The final way is chaotic maps can also be used to optimize the parameters of a chaotic system itself [12]. Implementing the CMRUN can take all the above approaches, but this paper uses the approach of creating a perturbation to the current solutions.

From Section 2, the best solution (x_b) is essential in enhancing the global search space to find the optimal solution in the original RUN algorithm. The best and worst solutions obtained at each iteration determine the searching mechanism of the RUN algorithm. Ahmadianfar et al. [3] used the rand parameter to introduce randomness and enhance the exploration search. Furthermore, Equations (13)–(15) provide search diversification to avoid local optima traps.

The range between adjacent positions:

$$\Delta x = 2 \times \text{rand} \times |\text{Stp}|. \quad (40)$$

The step size:

$$\text{Stp} = \text{rand} \times ((x_b - \text{rand} \times x_{\text{avg}}) + \gamma). \quad (41)$$

The scale factor:

$$\gamma = \text{rand} \times (x_n - \text{rand} \times (u - l)) \times \exp\left(-4 \times \frac{i}{\text{Maxi}}\right). \quad (42)$$

By their contribution to the global search of the RUN algorithm, these equations were selected to be improved by chaos, thus improving the overall performance of the original RUN.

The 10 chaotic maps used are represented as a vector, and each map is selected by inputting its index value as in Table 1. The initial value chosen for all the maps is 0.7. The final values of all the maps should lie in the range [0, 1], and thus all values in $[-1, 1]$ are normalized so that they are within the acceptable [0, 1] range. Usually, chaotic parameters are used in the parts of algorithms that require random parameters. In this paper, the rand parameter for Equations (40)–(42) was replaced with a chaotic parameter cp.

The chaos vector is first calculated, and then the index of the chaotic map to be integrated determines which vector values will be selected to introduce chaos in the RUN algorithm. The parameters selected are improved at each iteration to ensure new random values are produced with each iteration. These values are then used to introduce chaos in the RUN through Equations (40)–(42), which are modified as follows:

The range between adjacent positions is as follows:

$$\Delta x = 2 \times cp \times |Stp|. \quad (43)$$

The step size is as follows:

$$Stp = cp \times ((x_b - cp \times x_{avg}) + \gamma). \quad (44)$$

The scale factor is as follows:

$$\gamma = cp \times (x_n - cp \times (u - l)) \times \exp\left(-4 \times \frac{i}{Maxi}\right), \quad (45)$$

where x_n is the current position at each iteration.

The above parameters are used in the searching mechanism, and thus, they enhance the quality of solutions by increasing the search space. Furthermore, the algorithm begins optimization using a set of random initial solutions, which get updated after each iteration using the RK method, which employs the searching mechanism. This implies that the solutions after each iteration have chaotic behavior, and thus, the global search is enhanced. Equation (30) is further improved using the chaotic maps to enhance the exploration and exploitation phases by replacing r and with cp , as shown in Equation (46)

The same is done in the code's ESQ section, as shown in Equation (47).

$$\begin{aligned} &\text{if } cp < 0.5 \\ &\quad (\text{exploration phase}) \\ &\quad x_{n+1} = (x_c + r \cdot SF \cdot g \cdot x_c) + SF \cdot SM + \mu \cdot \text{randn} \\ &\quad \cdot (x_m - x_c) \\ &\text{else} \\ &\quad (\text{exploitation phase}) \\ &\quad x_{n+1} = (x_m + r \cdot SF \cdot g \cdot x_m) + SF \cdot SM + \mu \cdot \text{randn} \\ &\quad \cdot (x_{r1} - x_{r2}) \\ &\text{end} \end{aligned} \quad (46)$$

$$\begin{aligned} &\text{if } cp < 0.5 \\ &\quad \text{if } w < 1 \\ &\quad \quad x_{new2} = x_{new1} + r \cdot w \cdot |(x_{new1} - x_{avg}) + \text{randn}| \\ &\quad \text{else} \\ &\quad \quad x_{new2} = (x_{new1} - x_{avg}) + r \cdot w \cdot |(u \cdot x_{new1} - x_{avg}) + \text{randn}| \\ &\quad \text{end} \\ &\text{end.} \end{aligned} \quad (47)$$

The original RUN is modified to utilize chaos in updating solutions and converging to the global optimum solution by performing the above changes. Table 2 shows the pseudocode for the CMRUN algorithm.

4.1.2. Generating Pareto Fronts for the CMRUN. The original RUN and the chaotic RUN described above can only

Part 1. Initialization

Define the number of objective functions ($nObj = 2$)

Initialize the fitness function for both objective functions

Randomly generate the initial population for the CMRUN

Evaluate the objective function values of each population member

Sort the costs obtained from the objective function

Initialize the chaos parameters

Update the convergence curves of both objectives with the first best costs

Part 2. CMRUN operations

for $it = 1: MaxIt$

 Update the chaotic parameters

for $n = 1: N$

 Apply chaotic parameters in updating the algorithm's equations

 Determine the solutions x_w , x_b , and x_{best} for each objective function

 Perform operations to improve and update the solutions

 Update best costs for the objective functions

 Check if solutions go outside the search space and bring them back

 Update chaos parameters for ESQ

Enhance the solution quality

for $j = 1: dim$

 Determine $x_{n+1,j}$ from Equation 30

end for

 Perform boundary check for solutions again

if

 Evaluate position

if $f(x_n) < f(x_{new2})$

if $\text{rand} <$

 Determine position x_{new3}

end

end

end

 Modernize positions x_w and x_b

end for

 Modernize position

$it = it + 1$

end

Part 3. Return x_{best} and best costs

 Update Convergence Curves

ALGORITHM 2: The pseudocode of CMRUN

optimize single-objective problems. The first step to making the chaotic RUN multi-objective is modifying the objective function to take more than one objective. In this case, the objective function was modified to take two objectives and generate a set of solutions for each objective. The fitness function was also modified to record the solutions for two objectives, generating two convergence curves that form the Pareto Front [5].

(1) *The Pseudocode for the CMRUN algorithm.*

TABLE 2: Benchmark test functions.

Function	Name	Type	Dimension	Range
F1	SCHAFER	U	30	$[-100, 100]$
F2	POLONI	U	30	$[-100, 100]$
F3	VIENNET2	U	30	$[-4, 4]$
F4	VIENNET3	U	30	$[-3, 3]$
F5	TANAKA	U	30	$[0, \pi]$
F6	ZDT2	M	30	$[-100, 100]$
F7	ZDT3	M	30	$[-100, 100]$
F8	ZDT4	M	30	$[0, 1]$
F9	ZDT5	M	30	$[0, 1]$
F10	ZDT6	M	30	$[0, 1]$
F11	DTLZ1	M	30	$[-100, 100]$
F12	DTLZ2	M	30	$[-100, 100]$
F13	DTLZ4	M	30	$[-100, 100]$
F14	KURSAWE	M	30	$[-5, 5]$
F15	FONSECA-FLEMING	M	30	$[-4, 4]$
U-Unimodal			M-Multimodal	

4.2. *Evaluation of the Algorithm's Performance.* The 10 chaotic maps were integrated one by one into the CMRUN, and their performances were evaluated using 15 selected benchmark functions. The results of the best version of the CMRUN were compared with those of 11 known multi-objective optimization algorithms.

The benchmark test functions are of two kinds [3]:

- (i) *Unimodal Functions:* have one global optimum and test for an algorithm's exploitation and convergence to the global optimum.
- (ii) *Multimodal Functions:* have multiple local optima and are used to test for an algorithm's exploration and ability to avoid premature convergence.

The following parameters were kept constant to test the CMRUN's performance and the other 11 optimizers.

- (i) Population number = 50
- (ii) Dimension = 30
- (iii) Maximum iterations = 200
- (iv) Number of runs = 10.

The benchmark functions used are given in Table 2. The following 11 algorithms were tested to compare their performance with the CMRUN:

- (1) Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D) [20]
- (2) Multi-Objective Particle Swarm Optimization (MOPSO) [21]
- (3) Non-dominated Sorting Genetic Algorithm II (NSGA-II) [22]
- (4) Multi-Objective Firefly Algorithm (MOFA) [23]
- (5) Multi-Objective Bat Algorithm (MOBA) [24]
- (6) Strength Pareto Evolutionary Algorithm 2 (SPEA2) [25]

TABLE 3: Amplifier circuit parameters.

Variable	Symbol	Range
Input resistor	R_{in}	$[5,000, 15,000]$
Collector resistor	R_c	$[500, 1,500]$
Load resistor	R_L	$[5,000, 15,000]$
Emitter resistor	R_e	$[500, 15,000]$
Lower frequency	f_L	$[10K, 50K]$
Upper frequency	f_H	$[1M, 20M]$

- (7) Multi-Objective Cuckoo Search (MOCS) [26]
- (8) Multi-Objective Flower Pollination Algorithm (MOFPA) [27]
- (9) Multi-Objective Mayfly Optimization Algorithm (MOMA) [28]
- (10) Hybrid NSGAI-MOPSO Algorithm [29]
- (11) Multi-Objective Non-Sorted Moth FLAME (MOMFO) (NSMFO) [30]

To evaluate the performance, the minimum score of each optimizer for each test function was recorded for 10 runs. These scores' averages were then calculated with their standard deviations as the criteria for measuring performance.

4.3. *Circuit Design.* In this paper, a simple circuit for a single-stage amplifier was designed. As discussed in Section 3, a single-stage amplifier has many practical applications in the real world. This paper proposes an algorithm that could be used to optimize its circuit parameters. The circuit chosen is simple to test the CMRUN's convergence capability. Optimizing a simple circuit and giving out the best parameters would mean the algorithm is powerful enough to be applied to more complex circuit design problems.

The objective functions chosen in this paper are:

- (i) Noise
- (ii) Distortion

Noise is any unwanted signal that arises at the output of a circuit. Noise reduces the performance of an electronic device due to errors and reduced sensitivity. Distortion, on the other hand, is when the output signal has been changed by nonlinearity; hence, the circuit does not produce an output consistent with the input.

For the circuit selected, the parameters to be optimized are shown in Table 3.

From the values of resistances and frequencies obtained, the values of capacitances can easily be calculated from the standard formulae. This part of the paper aims to minimize noise and distortion and obtain the values of frequencies and resistances where the two objective functions are minimum and, hence, find the optimum performance of the circuit from the given ranges of inputs.

5. Results, Analysis, and Discussion

An HP Probook G3 with 8 GB RAM and Intel(R) Core(TM) i5-6200U CPU @ 2.30 GHz 2.40 GHz processor was used to run and test the algorithms using MATLAB R2021a.

TABLE 4: Comparison results of the 10 chaotic maps using 15 benchmark test functions.

FUNCTION	Chebyshev	Circle	Gauss	Iterative	Logistic	Piecewise	Sine	Singer	Sinusoidal	Tent
F1	2.14E-03	2.70E-04	1.14E-03	2.00E-05	7.16E-03	4.20E-04	5.00E-04	4.03E-03	4.03E-03	8.10E-04
F2	-4.00E+07	-4.00E+07	-4.00E+07	4.00E+07	4.00E+07	4.00E+07	-4.00E+07	-4.00E+07	-4.00E+07	-4.00E+07
F3	-1.40E+01	-1.40E+01	-1.40E+01	-1.40E+01	-1.40E+01	-1.40E+01	-1.40E+01	-1.40E+01	-1.40E+01	-1.39E+01
F4	1.52E+01	1.50E+01	1.50E+01	1.51E+01	1.52E+01	1.51E+01	1.52E+01	1.52E+01	1.51E+01	1.52E+01
F5	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F6	-7.89E+03	-6.41E+03	-4.39E+03	-2.79E+03	-2.84E+03	-4.14E+03	-3.27E+03	-2.58E+03	-4.28E+03	-6.13E+03
F7	-1.55E+02	-1.39E+02	-1.59E+02	-1.37E+02	-1.12E+02	-9.79E+01	-1.28E+02	-1.11E+02	-1.32E+02	-1.02E+02
F8	1.85E+02	2.01E+02	1.68E+02	2.06E+02	2.03E+02	2.06E+02	2.06E+02	2.03E+02	1.90E+02	2.02E+02
F9	7.15E-01	6.36E-01	8.48E-01	7.24E-01	6.81E-01	8.07E-01	6.05E-01	7.67E-01	6.55E-01	6.79E-01
F10	8.26E+00	8.25E+00	8.22E+00	8.27E+00	8.21E+00	8.25E+00	8.17E+00	8.29E+00	8.25E+00	8.36E+00
F11	-1.06E+07	-1.04E+07	-1.06E+07	-1.05E+07	-1.08E+06	-1.09E+06	-1.03E+06	-1.03E+06	-1.05E+06	-1.09E+06
F12	-5.03E+05	-4.98E+05	-4.72E+05	-4.82E+05	-4.95E+05	-4.84E+05	-4.86E+05	-5.01E+05	-5.02E+05	-4.93E+05
F13	-4.96E+05	-4.80E+05	-4.82E+05	-4.73E+05	-4.81E+05	-4.80E+05	-4.81E+05	-4.90E+05	-5.01E+05	-4.87E+05
F14	-3.70E+01	-3.71E+01	-3.70E+01	-3.68E+01	-3.65E+01	-3.65E+01	-3.59E+01	3.64E+01	-3.60E+01	3.68E+01
F15	-6.10E-04	-6.10E-04	-1.47E-03	-5.00E-04	-7.00E-04	-2.50E-04	-4.70E-04	-1.07E-03	1.00E-02	-5.80E-04

Bold values are the best results obtained for each function.

TABLE 5: Statistical results of CMRUN and 11 other algorithms using 15 benchmark functions.

FUNCTION	CMRUN	MOEAD	MOPSO	NSGA-II	MOFA	MOBA	SPEA-2	MOCs	MOFPA	MOMA	NSGA-II/MOPSO	NSMFO
F1	avg	2.14E-03	1.75E+01	1.18E-04	1.77E-09	0.00E+00	6.39E+00	9.82E-40	0.00E+00	9.22E-10	2.05E-06	9.00E-14
	std	2.89E-03	5.23E+01	1.38E-04	5.27E-09	0.00E+00	6.42E+00	2.94E-39	0.00E+00	2.76E-09	6.14E-06	2.70E-13
F2	avg	-4.00E+07	-3.61E+07	-4.00E+07	-2.86E+08	-4.00E+07	-2.38E+07	-3.97E+07	-4.00E+07	-4.00E+07	-4.00E+07	-4.00E+07
	std	5.48E+01	4.41E+06	9.50E+04	8.48E+07	0.00E+00	5.94E+06	4.07E+05	1.66E-01	1.66E-01	4.26E-01	0.00E+00
F3	avg	-1.40E+01	-1.39E+01	-1.40E+01	-1.40E+01	-1.40E+01	-1.39E+01	-1.40E+01	-1.40E+01	-1.40E+01	-1.40E+01	-3.80E+00
	std	2.90E-02	8.64E-02	4.37E-02	2.22E-03	0.00E+00	7.09E-02	2.00E-03	0.00E+00	0.00E+00	1.75E-04	1.56E+01
F4	avg	1.52E+01	1.60E+01	1.52E+01	1.50E+01	1.50E+01	1.53E+01	1.50E+01	1.50E+01	1.50E+01	1.50E+01	1.50E+01
	std	2.19E-01	3.67E-01	2.56E-01	8.40E-03	0.00E+00	1.92E-01	3.03E-02	6.00E-05	2.52E-04	2.38E-04	1.85E-04
F5	avg	0.00E+00	1.06E-03	2.56E-03	-1.08E+00	0.00E+00	1.34E-01	0.00E+00	0.00E+00	1.00E-05	3.67E-18	6.74E-15
	std	0.00E+00	3.19E-03	5.66E-03	1.39E-02	0.00E+00	5.78E-02	0.00E+00	0.00E+00	3.00E-05	5.61E-18	6.74E-15
F6	avg	-7.89E+03	-5.34E+05	-2.85E+11	-1.15E+10	-4.84E+10	-2.23E+03	-1.22E+07	-2.42E+19	-1.42E+11	-3.37E+06	-4.92E+18
	std	6.35E+03	1.52E+06	6.46E+11	1.82E+10	1.37E+11	2.14E+03	2.86E+07	2.09E+19	2.00E+11	9.88E+06	1.34E+19
F7	avg	-1.55E+02	-1.44E+02	-6.06E+02	-6.90E+02	-4.30E+02	-6.25E-02	-6.35E+02	-5.93E+02	-5.50E+02	-6.99E+02	-5.99E+02
	std	1.15E+02	6.55E+01	9.45E+01	2.44E+02	1.45E+01	4.97E+00	4.64E+01	2.63E+00	2.49E+01	1.14E-13	2.45E-02
F8	avg	1.85E+02	8.83E+01	1.68E+01	NA	1.11E+01	2.08E+02	5.70E+01	1.10E+01	1.23E+01	0.00E+00	1.22E+01
	std	6.19E+01	2.35E+01	8.85E+00	NA	1.53E+00	1.93E+01	1.32E+01	2.34E+00	1.86E+00	0.00E+00	6.98E+00
F9	avg	7.15E-01	1.87E+00	6.58E+00	2.39E+00	-7.17E-01	7.59E+00	1.91E-01	-4.73E+00	-2.45E+00	1.00E+00	-6.22E+00
	std	6.74E-02	9.78E-01	2.58E+00	1.01E+00	8.59E-01	8.93E-01	8.75E-01	1.56E+00	4.77E-01	0.00E+00	1.05E-01
F10	avg	8.26E+00	6.75E+00	4.88E+00	-1.16E+02	2.32E+00	8.32E+00	4.97E+00	5.28E+00	2.06E+00	2.81E-01	2.81E-01
	std	1.38E-01	3.59E-01	1.30E+00	2.99E+02	1.91E-01	1.91E-01	2.01E-01	1.32E+00	1.30E+00	0.00E+00	5.55E-17
F11	avg	-1.06E+07	-1.00E+07	-1.04E+07	-3.36E+07	-2.09E+07	-1.07E+07	-1.96E+07	-2.88E+07	-1.76E+07	-1.00E+06	-2.81E+07
	std	8.85E+05	1.13E+06	3.47E+06	4.45E+06	7.61E+05	7.93E+05	8.58E+05	5.34E+04	7.99E+05	4.89E+03	7.07E+05
F12	avg	-5.03E+05	-2.38E+05	-5.24E+05	-1.04E+06	-4.73E+05	-1.81E+05	-4.87E+05	-5.77E+05	-4.93E+05	-1.99E+04	-5.69E+05
	std	1.62E+04	2.33E+04	2.92E+04	7.51E+04	1.30E+04	6.01E+04	1.36E+04	4.77E+03	2.40E+04	1.38E+02	1.03E+04
F13	avg	-4.96E+05	-2.38E+05	-5.59E+05	-1.17E+06	-5.01E+05	-2.30E+05	-5.12E+05	-5.79E+05	-5.42E+05	-2.02E+04	-5.77E+05
	std	8.87E+03	2.33E+04	1.15E+04	9.96E+04	8.89E+03	1.24E+04	7.79E+03	7.01E+02	1.54E+04	9.47E-03	9.17E+03
F14	avg	-3.70E+01	-2.87E+01	-3.00E+01	-4.76E+01	-4.77E+01	-3.51E+01	-4.57E+01	-4.76E+01	-4.74E+01	NA	-4.77E+01
	std	1.97E+00	1.38E+00	1.86E+00	2.68E-02	5.62E-02	1.98E+00	3.78E-01	1.70E-01	2.19E-01	NA	2.01E-02
F15	avg	-6.10E-04	1.70E-02	8.80E-03	-2.04E-03	-2.80E-03	4.64E-01	-2.05E-03	-2.80E-03	-2.68E-03	-2.63E-03	-2.72E-03
	std	2.22E-03	2.68E-02	1.38E-02	1.70E-03	0.00E+00	4.09E-01	2.28E-04	0.00E+00	3.83E-05	1.47E-04	9.80E-05

Bold values are the best results obtained for each function to show the algorithms that performs best.

TABLE 6: CMRUN performance rank.

Function	Rank
F1	10
F2	2
F3	1
F4	9
F5	2
F6	11
F7	10
F8	10
F9	7
F10	11
F11	9
F12	6
F13	9
F14	8
F15	1

5.1. Benchmark Test Results of the Chaotic Multi-Objective Runge–Kutta Optimization Algorithm. The CMRUN was implemented using 10 chaotic maps and ran 10 times for each benchmark function. The mean scores were recorded to determine the best chaotic map out of the 10. The best CMRUN version was selected for comparison with the other 11 multi-objective algorithms. The other algorithms were also run 10 times, and their average scores and standard deviations were recorded. The benchmark functions are categorized into two:

- (i) Unimodal test functions (F1–F5)
- (ii) Multimodal test functions (F6–F15)

The comparison results of the chaotic maps are shown in Table 4 (the bold numbers are the best results obtained for each function). The results with lower values are the better results. The CMRUN with the Chebyshev map had the best averages for functions F2, F3, F5, F6, F11, F12, and F15. The CMRUN with the Circle map realized the best averages for functions F2, F3, F4, F5, F14, and F15. The CMRUN with the Gauss map best performed in functions F2, F3, F4, F5, F7, F8, and F11. The CMRUN with the Iterative map attained the best averages for functions F1, F2, F3, and F5. The CMRUN with the Logistic map, the Piecewise map, the Singer map, and the Tent map realized the best averages for functions F2, F3, and F5.

The CMRUN attained the best averages for functions F2, F3, F5, F9, and F10 for the Sine map. Lastly, for the Sinuisodal map, the CMRUN achieved the best scores for functions F2, F3, F5, and F13. All the maps realized the global minima for functions F2 and F5. For function F3, the CMRUN achieved global minima with all maps except the Tent map. From these results, the best chaotic maps for the CMRUN are the Chebyshev and Gauss maps, as they both reached the best scores for 7 out of the 15 benchmark functions. In this paper, the CMRUN with the Chebyshev map was chosen to compare with the 11 multi-objective algorithms.

From Table 5, the averages for each algorithm were compared for every benchmark function (the best results are in bold). In this paper, the CMRUN with the Chebyshev map is considered for comparison with the other algorithms.

For the unimodal functions, CMRUN's best performance is in F3, where it ranks joint first with other algorithms, and in F2 and F5, it ranks second. The CMRUN has the best performance and edges other algorithms in only one of the ten multimodal functions (F15). The CMRUN attains the global optimum in one benchmark function (F3) and its lowest standard deviation in function F5, which is zero.

An algorithm's convergence rate and solution quality should be considered to determine whether the algorithm is stuck in local optima. The CMRUN struggles in optimizing multimodal functions and ranks position 11 twice. This indicates the CMRUN has difficulty maintaining the balance between exploration and exploitation. The success of an algorithm is linked to its balance between the two phases.

Exploitation focuses on utilizing the current knowledge by concentrating the algorithm's search around areas likely to contain the best solutions. It refines the search around high-quality solutions and hence finds the local optimum. On the other hand, exploration ventures toward unexplored areas in the search space to find potentially better solutions, thus maintaining diversity. As the optimization progresses, exploration finds potential global solutions. Then, the search space focus shifts toward exploitation, which refines and improves these solutions to find the best global solutions.

The unimodal functions test for an algorithm's exploitative behavior, and the CMRUN shows good exploitation capabilities from the results. The multimodal functions test an algorithm's explorative behavior, and the CMRUN struggles to explore the global search space depending on the problem.

From the comparison of the average scores, the CMRUN ranks as shown in Table 6.

5.2. Optimized Circuit Design. The CMRUN's performance in optimizing circuit design was compared to the performance of the other 11 algorithms. The following parameters were kept constant to conduct this part of the paper.

- (i) Population number = 100
- (ii) Dimension = 50
- (iii) Maximum iterations = 1,000
- (iv) Number of runs = 10

Table 7 shows the comparison results of optimizing circuit parameters for all 10 chaotic maps. The CMRUN with the iterative map achieves the lowest noise value, while the CMRUN with the Gauss map realizes the highest noise value. The maps rank as follows: Iterative, Logistic, Chebyshev, Sinuisodal, Sine, Tent, Singer, Circle, Piecewise, and Gauss.

The CMRUN with the circle map attains the lowest distortion value, while the CMRUN with the logistic map has the highest. The maps rank as follows: Circle, Sinuisodal, Sine, Tent, Singer, Chebyshev, Piecewise, Iterative, Gauss, and Logistic. The CMRUN with the iterative map offers

TABLE 7: Comparison results of the 10 chaotic maps for optimized circuit design.

OBJFNC	Chebyshev	Circle	Gauss	Iterative	Logistic	Piecewise	Sine	Singer	Sinusoidal	Tent
Noise	227.202	227.3447	227.4031	227.1816	227.1996	227.3783	227.2442	227.3169	227.2086	227.2878
Distortion	0.0508	0.0003	0.0539	0.0525	0.0552	0.0515	0.0468	0.0487	0.0463	0.0479
Pareto sum	227.2528	227.345	227.457	227.2341	227.2548	227.4298	227.291	227.3656	227.2549	227.3357

Bold values are the best results (lowest) obtained for circuit optimization.

TABLE 8: Results for optimized circuit design.

OBJ/FNC	CMRUN	MOEAD	MOPSO	NSGA-II	MOFA	MOBA	SPEA-2	MOCS	MOFPA	MOMA	NSGA-II/MOPSO	NSMFO
Noise	227.202	227.507	258.2846	227.9789	227.6274	235.7833	227.763	227.9251	227.7534	227.2536	227.4659	229.0733
Distortion	0.0508	0.07429	0.004399	0.079716	0.0002	0.0002	0.000193	0.0002	0.0002	0.056177	0.000156	0.0002
Pareto sum	227.2528	227.5813	258.289	228.0586	227.6276	235.7835	227.7632	227.9253	227.7536	227.3097	227.4661	229.0735

Bold values are the best results (lowest) obtained for circuit optimization.

TABLE 9: Optimized design variables.

Variable	Symbol	Value
Input resistor	R_{in}	11,300
Collector resistor	R_c	800
Load resistor	R_L	5,150
Emitter resistor	R_e	8,740
Lower frequency	f_L	20,600
Upper frequency	f_H	14,400,000

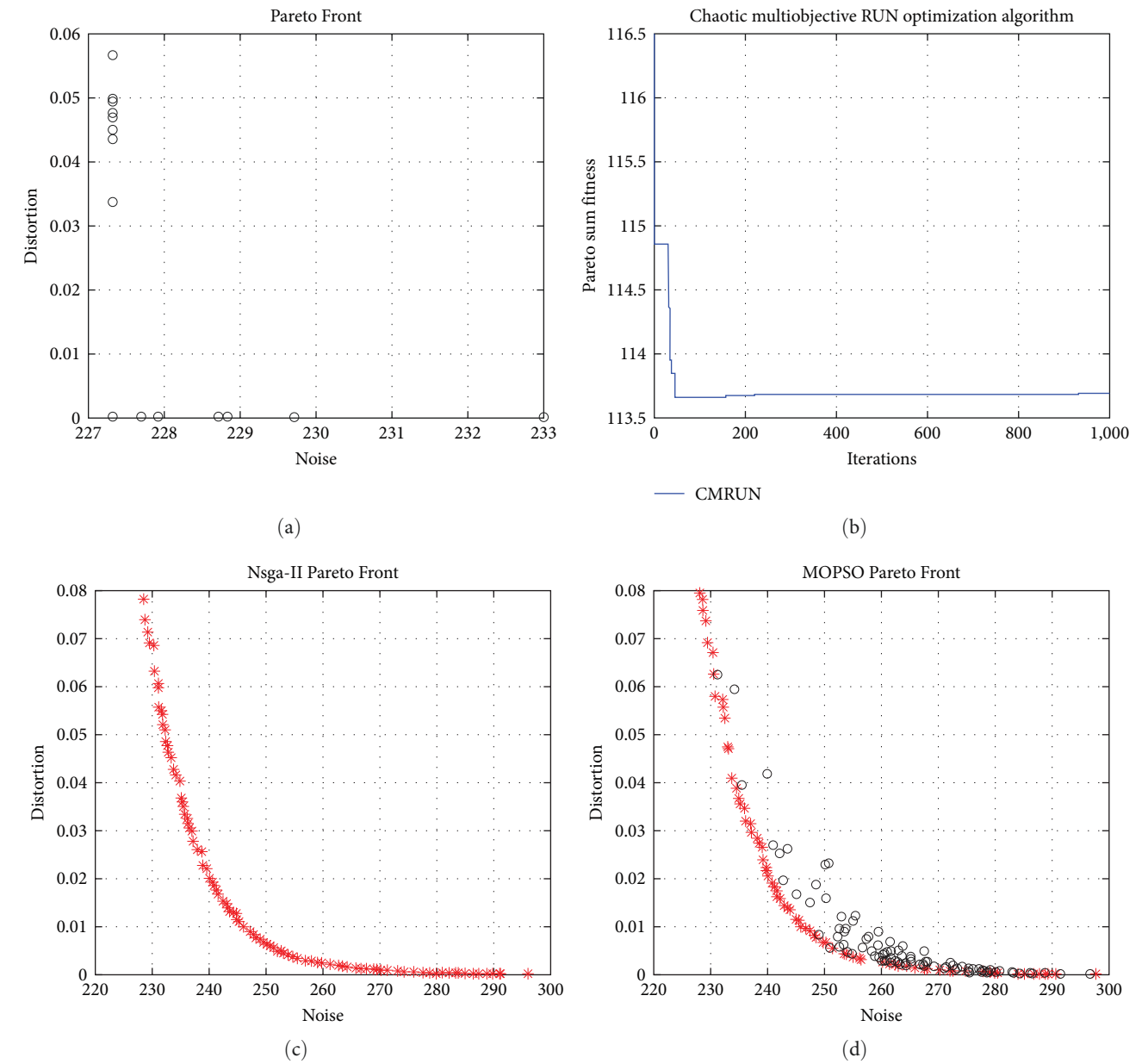
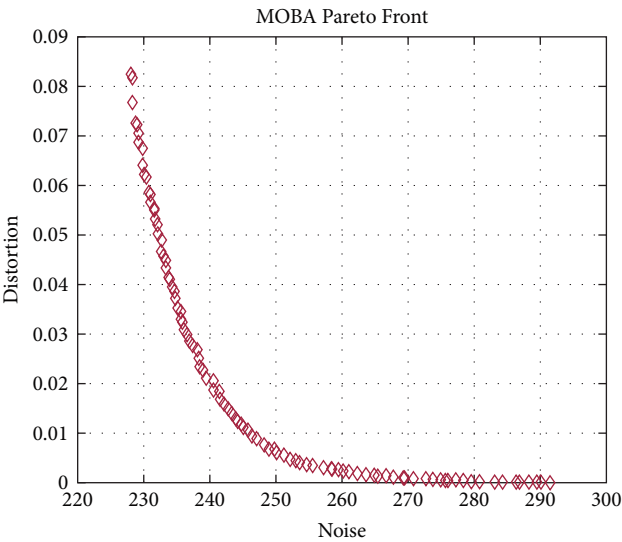
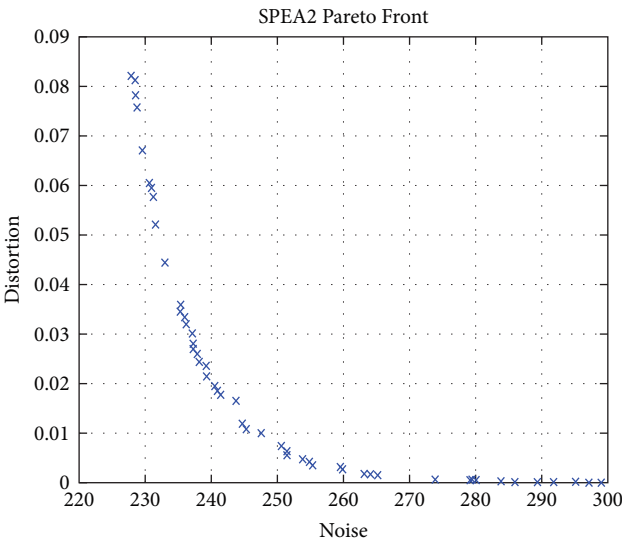


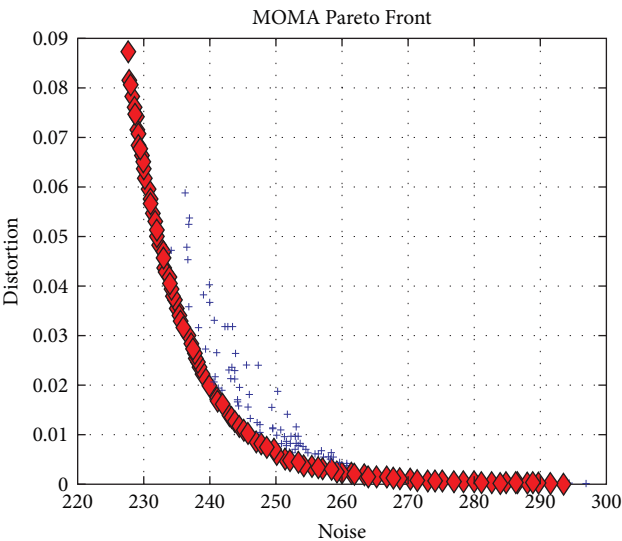
FIGURE 8: Continued.



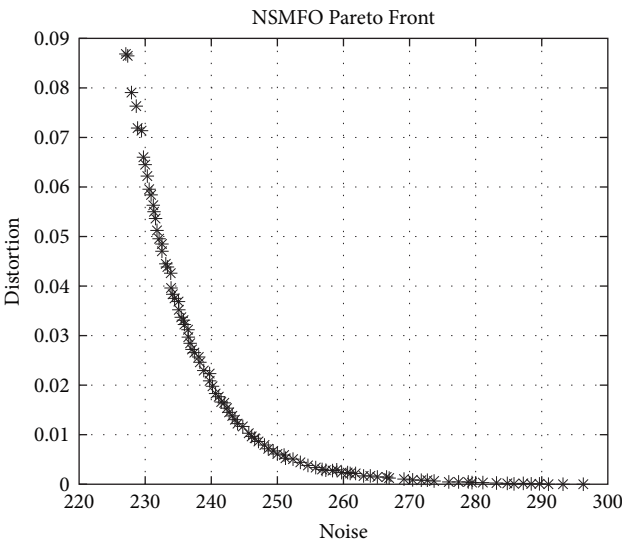
(e)



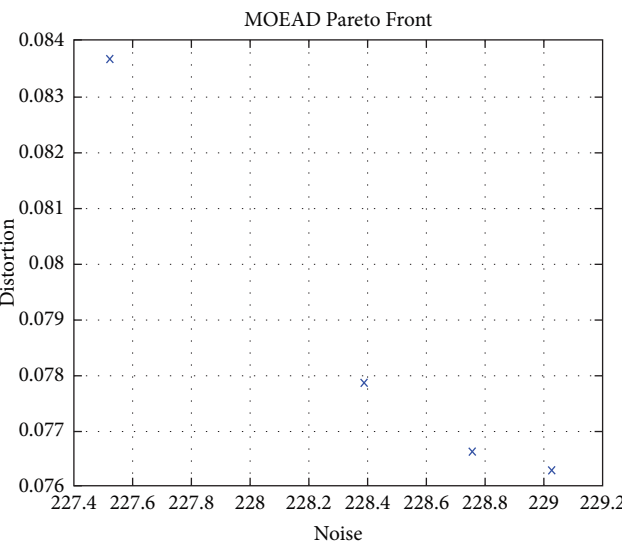
(f)



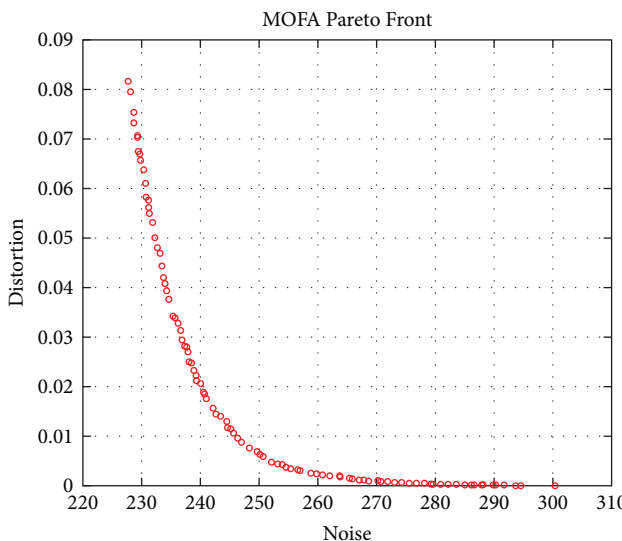
(g)



(h)



(i)



(j)

FIGURE 8: Continued.

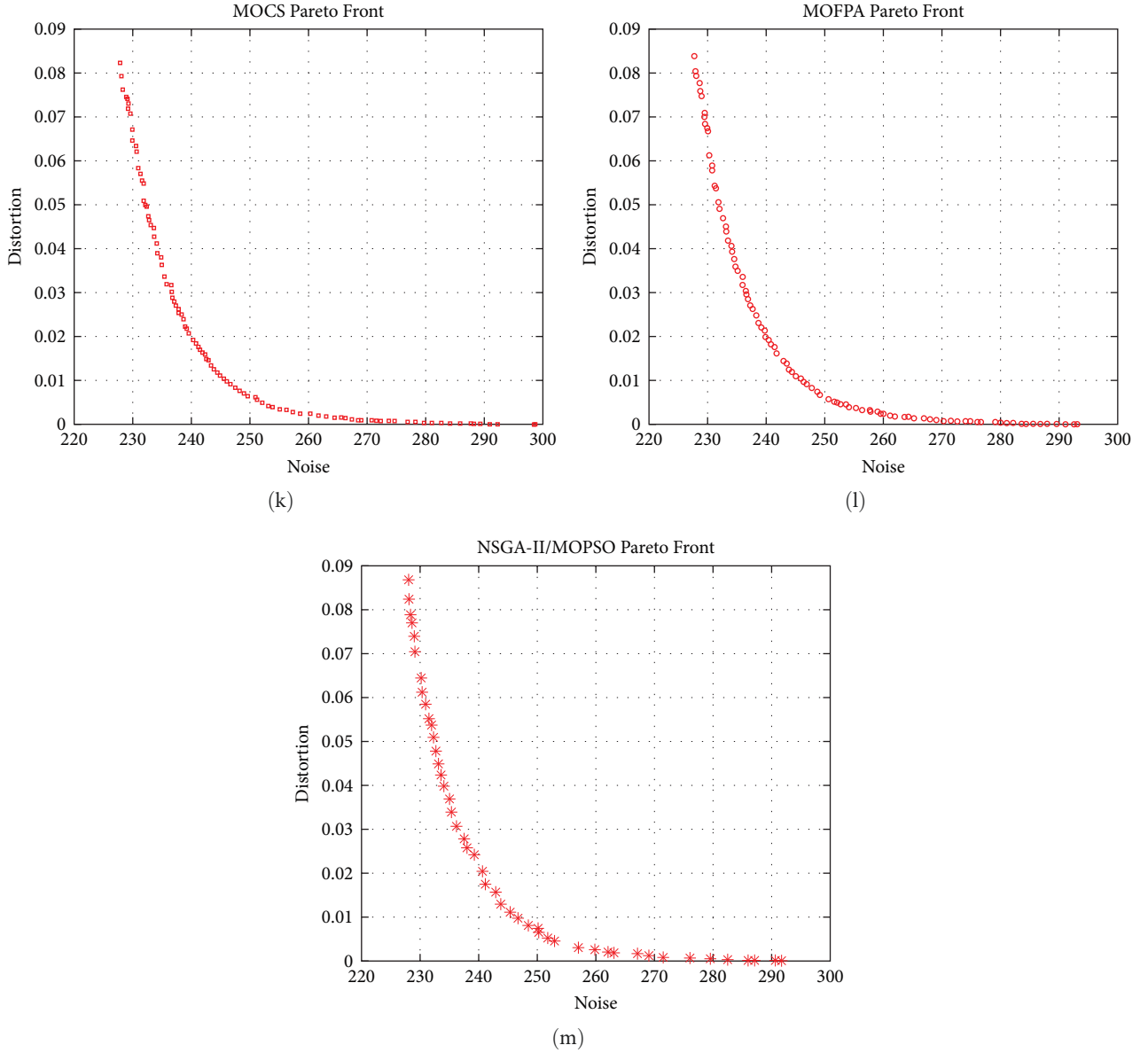


FIGURE 8: (a) CMRUN, (b) CMRUN Pareto fitness plot, (c) NSGA-II, (d) MOPSO, (e) MOBA, (f) SPEA-2, (g) MOMA, (h) NSMFO, (i) MOEAD, (j) MOFA, (k) MOCS, (l) MOFPA, and (m) NSGA-II/MOPSO.

the best performance as it provides the lowest Pareto sum for all the maps, showing a balanced performance between the two objectives.

Table 8 shows the results for optimized circuit design parameters of the CMRUN map compared to the other 11 algorithms. For consistency, the results of the CMRUN with the Chebyshev map were chosen for comparison. From the results, the CMRUN outperforms the other algorithms in optimizing the circuit parameters. It has the lowest Pareto sum fitness of all the algorithms with weights of $[0.5, 0.5]$ for the objective functions.

The equal weights mean the objective functions are considered equally important. From the results, CMRUN has the lowest noise value of 227.202, followed closely by MOMA. The other algorithms have slightly higher noise values, with

MOPSO having the highest noise value. For the distortion values, the algorithms MOFA, MOBA, SPEA-2, MOCS, MOFPA, NSGA-II/MOPSO, and NSMFO achieved the lowest distortion values of 0.0002 or less. CMRUN has a slightly higher value for distortion, the sixth highest. Based on the Pareto sum, CMRUN has the lowest Pareto sum, followed closely by MOMA. Therefore, CMRUN performs generally well compared to other algorithms despite having a higher distortion value than the others because it offers the lowest noise value and lowest Pareto sum, showing better balance in optimizing both objectives.

The CMRUN's performance in circuit optimization indicates that it performs well in minimizing the fluctuations in the noise values. Still, there are deviations in achieving the lowest value possible for distortion values. This might be due

to conflicts or tradeoffs between distortion and noise. Although the CMRUN shows excellence in reducing noise, it sacrifices distortion as improving noise performance deteriorates distortion. Overall, the performance of CMRUN in circuit optimization is superior as it defeats all the algorithms in Pareto sum and offers the lowest noise value. The individual circuit parameters were also recorded and tabulated.

The best parameters obtained are shown in the Table 9.

The convergence curves for CMRUN and the 11 algorithms are shown in Figure 8. These curves show the algorithms' convergence rate when optimizing circuit parameters.

The convergence curves show that the CMRUN has a very fast convergence rate. This indicates the algorithm's superiority in obtaining optimal solutions using few iterations. The CMRUN outperforms the other algorithms in search exploration, quickly narrowing the feasible region and converging to the global solution when optimizing circuit parameters. Although a fast convergence could also indicate the algorithm has converged prematurely, and the solutions are suboptimal, this is not the case for the CMRUN, according to the results.

Therefore, it is essential to consider both the convergence rate and the quality of solutions. From the solutions obtained by CMRUN, the solutions are of quality. Thus, its fast convergence rate is desirable, indicating that the algorithm requires fewer iterations to achieve a satisfactory solution. This can lead to significant time savings and computational efficiency, especially in complex optimization problems with a large search space or computationally expensive objective functions.

6. Conclusion

This paper proposed a CMRUN to optimize circuit design. Chaotic maps and Pareto Front were incorporated to enhance the RUN to improve its exploration capability and handle multiple objectives. The CMRUN excelled in exploitation but struggled in exploration searches when optimizing multimodal functions.

From the results, the CMRUN is susceptible to getting stuck in local optima and thus converges prematurely, especially when optimizing multimodal functions. It shows it may not perform well for some circuit design optimization problems. This work can further be extended by implementing standard operators in the CMRUN, such as levy walks (LWs), crossover operators, mutation operators, or the opposite-based learning method to better the solution quality [3].

Further research should be done to develop a more practical implementation of the CMRUN that can optimize complex problems, especially multimodal ones. Also, it would be noteworthy to test whether other chaotic mapping strategies offer better randomization and, thus, a better balance between the exploitation and exploration. The CMRUN could further be upgraded to handle many-objective optimization problems.

Data Availability

The data used to derive the results of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was sanctioned and fully funded by the University of Nairobi, Department of Electrical and Information Engineering.

References

- [1] M. W. Cohen, M. Aga, and T. Weinberg, "Genetic algorithm software system for analog circuit design," *Procedia CIRP*, vol. 36, pp. 17–22, 2015.
- [2] L. W. Jer, G. L. J. Xiong, S. C. Neoh, and A. Marzuki, "GA-based optimization for circuit design assistance," in *2012 Third International Conference on Intelligent Systems Modelling and Simulation*, pp. 732–736, IEEE, Kota Kinabalu, Malaysia, 2012.
- [3] I. Ahmadianfar, A. A. Heidari, A. H. Gandomi, X. Chu, and H. Chen, "RUN beyond the metaphor: an efficient optimization algorithm based on Runge Kutta method," *Expert Systems with Applications*, vol. 181, Article ID 115079, 2021.
- [4] X. Shu, Y. Liu, J. Liu, M. Yang, and Q. Zhang, "Multi-objective particle swarm optimization with dynamic population size," *Journal of Computational Design and Engineering*, vol. 10, no. 1, pp. 466–467, 2023.
- [5] B. S. Yildiz, P. Mehta, N. Panagant, S. Mirjalili, and A. R. Yildiz, "A novel chaotic Runge Kutta optimization algorithm for solving constrained engineering problems," *Journal of Computational Design and Engineering*, vol. 9, no. 6, pp. 2452–2465, 2022.
- [6] A. L. Jaimes, S. Z. Martínez, and C. A. C. Coello, "An introduction to multi-objective optimization techniques," in *Optimization in Polymer Processing*, pp. 29–57, Nova Science Publishers, 2011.
- [7] I. Ahmadianfar, B. Halder, S. Heddami et al., "An enhanced multioperator Runge–Kutta algorithm for optimizing complex water engineering problems," *Sustainability*, vol. 15, no. 3, Article ID 1825, 2023.
- [8] B. Benhala, "Hybridization approaches of metaheuristics for optimal analog circuit design," *International Journal of Microwave and Optical Technology*, vol. 9, no. 6, pp. 421–428, 2014.
- [9] E. Cengiz, C. Yilmaz, H. Kahraman, and Ç. Sulçmez, "Improved Runge Kutta optimizer with fitness distance balance-based guiding mechanism for global optimization of high-dimensional problems," *Duzce University Journal of Science and Technology*, vol. 9, no. 6, pp. 135–149, 2021.
- [10] R. M. Devi, M. Premkumar, P. Jangir, M. A. Elkotb, R. M. Elavarasan, and K. S. Nisar, "IRKO: an improved Runge–Kutta optimization algorithm for global optimization problems," *Computers, Materials & Continua*, vol. 70, no. 3, pp. 4803–4827, 2022.
- [11] N. Aslimani, T. El-ghazali, and R. Ellaia, "A new chaotic-based approach for multi-objective optimization," *Algorithms*, vol. 13, no. 9, Article ID 204, 2020.
- [12] H. Gezici and H. Livatyali, "Chaotic Harris Hawks optimization algorithm," *Journal of Computational Design and Engineering*, vol. 9, no. 1, pp. 216–245, 2022.
- [13] A. R. Jordehi, "Chaotic bat swarm optimisation (CBSO)," *Applied Soft Computing*, vol. 26, pp. 523–530, 2015.
- [14] M. Kohli and S. Arora, "Chaotic grey wolf optimization algorithm for constrained optimization problems," *Journal of*

- Computational Design and Engineering*, vol. 5, no. 4, pp. 458–472, 2018.
- [15] S. Saremi, S. Mirjalili, and A. Lewis, “Biogeography-based optimisation with chaos,” *Neural Computing and Applications*, vol. 25, no. 5, pp. 1077–1097, 2014.
 - [16] D. Dhawale, V. K. Kamboj, and P. Anand, “An improved chaotic Harris Hawks optimizer for solving numerical and engineering optimization problems,” *Engineering with Computers*, vol. 39, pp. 1183–1228, 2023.
 - [17] M. Dachyar, Farizal, and C. Kurniajaya, “Using genetic algorithm to generate Pareto-Front in multi-objective problem,” *International Journal of Computer Applications*, vol. 180, no. 51, pp. 21–25, 2018.
 - [18] J. Knowles and D. Corne, “The Pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimisation,” in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, vol. 1, pp. 98–105, IEEE, Washington, DC, USA, 1999.
 - [19] SASMITA, “Single stage transistor amplifier,” *Electronics Post*, 2020.
 - [20] Y. M. Heris, “Multi-objective evolutionary algorithm based on decomposition (MOEA/D),” *MATLAB Central File Exchange*, 2023.
 - [21] Y. M. Heris, “Multi-objective particle swarm optimization (MOPSO),” *MATLAB Central File Exchange*, 2023.
 - [22] Y. M. Heris, “Non-dominated sorting genetic algorithm II (NSGA-II),” *MATLAB Central File Exchange*, 2023.
 - [23] X.-S. Yang, “Multiobjective firefly algorithm for continuous optimization,” *Engineering with Computers*, vol. 29, no. 2, pp. 175–184, 2013.
 - [24] X.-S. Yang, “Bat algorithm for multi-objective optimisation,” *International Journal of Bio-Inspired Computation*, vol. 3, no. 5, pp. 267–274, 2011.
 - [25] Y. M. Heris, “Strength Pareto Evolutionary Algorithm 2 (SPEA2),” *MATLAB Central File Exchange*, 2023.
 - [26] X.-S. Yang and S. Deb, “Multiobjective cuckoo search for design optimization,” *Computers & Operations Research*, vol. 40, no. 6, pp. 1616–1624, 2013.
 - [27] X.-S. Yang, M. Karamanoglu, and X. He, “Flower pollination algorithm: a novel approach for multiobjective optimization,” *Engineering Optimization*, vol. 46, no. 9, pp. 1222–1237, 2013.
 - [28] K. Zervoudakis and S. Tsafarakis, “A mayfly optimization algorithm,” *Computers & Industrial Engineering*, vol. 145, Article ID 106559, 2020.
 - [29] A. Sundaram, “Combined heat and power economic emission dispatch using hybrid NSGA II-MOPSO algorithm incorporating an effective constraint handling mechanism,” *IEEE Access*, vol. 8, pp. 13748–13768, 2020.
 - [30] D. P. Jangir, “Multi objective non sorted moth flame (MOMFO) (NSMFO) Matlab,” *MATLAB Central File Exchange*, 2023.