Wiley Interdisciplinary Reviews: Computational Statistics

WILEY

WIREs COMPUTATIONAL STATISTICS

ADVANCED REVIEW OPEN ACCESS

A Review of Benchmark and Test Functions for Global Optimization Algorithms and Metaheuristics

M. Z. Naser[1] | Mohammad Khaled Al-Bashiti[1] | Arash Teymori Gharah Tapeh[1] | Ahmad Naser[2] | Venkatesh Kodur[3] | Rami Hawileh[4] | Jamal Abdalla[4] | Nima Khodadadi[5] | Amir H. Gandomi[6,7,8] | Armin Dadras Eslamlou[9]

[1]Clemson University, Clemson, South Carolina, USA | [2]University of Manitoba, Winnipeg, Manitoba, Canada | [3]Michigan State University, East Lansing, Michigan, USA | [4]American University of Sharjah, Sharjah, UAE | [5]University of Miami, Coral Gables, Florida, USA | [6]University of Technology Sydney, Sydney, New South Wales, Australia | [7]Óbuda University, Budapest, Hungary | [8]Khazar University, Baku, Azerbaijan | [9]Glenn Department of Civil Engineering, Clemson University, Clemson, South Carolina, USA

Correspondence: M. Z. Naser (mznaser@clemson.edu)

Received: 22 January 2025 | **Revised:** 24 April 2025 | **Accepted:** 25 April 2025

Commissioning Editor: James E. Gentle | **Editor-in-Chief:** David W Scott

Funding: The authors received no specific funding for this work.

Keywords: artificial intelligence | optimization | test functions

ABSTRACT

Benchmarking in optimization is a critical step in evaluating the performance, robustness, and scalability of machine learning algorithms and metaheuristics. While trends in benchmark design continue to evolve, synthetic functions remain vital for fundamental stress tests and theoretical evaluations. As several benchmark and test functions have been developed and derived over the past decades, little attention has been given to classifying such test functions and the rationale behind their usage. From this lens, this paper reviews and categorizes a broad range of functions often employed in assessing optimizers and metaheuristics. More specifically, we classify test functions based on modality, dimensionality, separability, smoothness, constraints, and noise characteristics to offer a broad view that aids in selecting appropriate benchmarks for various algorithmic challenges. Then, this review also discusses in detail the 25 most commonly used functions in the open literature and proposes two new, highly dimensional, dynamic, and challenging functions that could be used for testing new algorithms. Finally, this review identifies gaps in current benchmarking practices and directions for future research, as well as suggests best practices and guidelines.

1 | Introduction

Machine learning (ML) algorithms often rely on iterative adjustments of parameter values to tune objective functions to address optimization tasks such as minimizing classification error, maximizing likelihood in statistical models (Sra et al. 2012). These tasks are fundamental in academic research and industrial applications (i.e., fine-tuning neural networks for large-scale image recognition and optimizing recommender system structure for personalized experiences). Similarly, metaheuristics (e.g., genetic algorithms and particle swarm optimization) can also tackle complex optimization problems that challenge classical gradient-based methods or lack analytically tractable structures (Boussaïd et al. 2013). While gradient descent and its variants dominate certain ML contexts, metaheuristics excel in domains marked by rugged fitness landscapes, multiple optima, or complex constraints. Therefore, the pursuit of efficient optimization is central to many processes in ML and metaheuristics (Sun et al. 2020).

[Correction added on 11 June 2025, after first online publication: The last name of the sixth author has been updated to Hawileh.]

Benchmark and test functions provide a controlled environment for assessing the efficacy and limitations of the aforementioned methods (Yang 2010). As without appropriate benchmarks, it is difficult to compare algorithms, identify strengths and weaknesses, or refine methodological assumptions (Fonseca et al. 2020). For instance, an algorithm that converges rapidly on one problem type might be ill-suited for other scenarios if its performance is only measured on simplistic benchmarks (Chopard and Tomassini 2018). Thus, the development and selection of representative test functions heavily influence the reported successes or failures of novel optimization strategies. More specifically, within ML research, optimization underlines multiple stages of algorithm development (i.e., parameter initialization, model architecture exploration), and hence, robust optimization routines are critical to achieving high performance. Equally, though sometimes viewed as blackbox methods, metaheuristics rely on thorough testing to validate their capabilities across varied landscapes and constraints (Omidvar et al. 2021; Omidvar et al. 2022).

Despite the recognized need for benchmarking, the field suffers from unstandardized and inconsistent test function usage (Longjohn et al. 2024; Bartz-Beielstein et al. 2010). For example, researchers frequently adopt a limited suite of well-known benchmarks, such as Sphere, Rastrigin, Rosenbrock, and Ackley functions, without systematically examining their relevance to newly proposed methods (White et al. 2013). This over-reliance on a narrow set of benchmarks risks reinforcing algorithmic biases and can lead to inflated performance claims (Runarsson and Yao 2005). Besides the traditional benchmark functions discussed herein, a configurable benchmark generator, called GNBG, permits on-demand adjustment of landscape features and thus enables the construction of tailored test suites for deep algorithmic analysis (Yazdani, Omidvar et al. 2023). Furthermore, traditional benchmarks often neglect dynamic or stochastic conditions, yet these properties are increasingly common in real-world scenarios. In addition, fairness and reproducibility in optimization studies hinge on the uniform adoption of recognized benchmarks. Therefore, there is a critical need to examine a broad range of functions to investigate attributes such as modality, dimensionality, separability, and noise or constraints to attain a comprehensive algorithmic evaluation (Beiranvand et al. 2017; Parejo et al. 2012).

This review addresses the abovementioned challenges through a structured approach to classifying and analyzing test and benchmark functions. Given the large landscape of this area, we focus this review primarily on synthetic functions (such as those designed purely for theoretical or illustrative purposes). Synthetic functions remain valuable for fundamental stress testing, as they allow precise control over landscape features like the location of optima or the degree of multimodality (Cheng et al. 2017; Okabe et al. 2004; Jamil et al. 2013). Therefore, our first objective is to systematically identify and categorize functions based on features such as modality (unimodal vs. multimodal), dimensionality (low-dimensional vs. high-dimensional), and the presence or absence of constraints. Our second objective is to highlight gaps in the current catalog of benchmark functions, which stem from several factors, including the lack of standardized and dynamic problems. Finally, our review proposes best practices for selecting test functions and identifies challenges and trajectories for future research as a means to encourage a more careful and consistent adoption of benchmarks across ML and metaheuristics communities.

2 | Background and Literature Review

Optimization benchmarking has been driven by the imperative to objectively compare diverse algorithms in controlled yet informative settings (Chinneck 2008). Early studies emphasized hand-crafted functions with explicit closed-form expressions as references against which new methods/algorithms could be tested (Ŝtefek 2011). These preliminary efforts established practices such as evaluating convergence speed and solution accuracy on a small set of widely recognized benchmarks (Tedford and Martins 2010). Although elementary by today's standards, these initial steps established fundamental themes: the need for transparency in function definitions and reproducible experimental setups, and to cover a broad range of problem features (Alimoradi et al. 2010).

A significant turning point emerged when developers recognized that too narrow an inventory of benchmarks risked biasing algorithm design and overstating performance (Brownlee 2007). In response, developers started documenting the risks of relying on single-problem comparisons or simplistic test suites that captured only smooth, low-dimensional landscapes (Sala and Muller 2020). In the 1990s and early 2000s, work introduced intricate functions with sharp local minima, deceptive global structures, or coupling among variables. Alongside these designs, researchers also proposed elaborate frameworks that categorized functions by dimensionality, modality, and separability (Weise et al. 2014). Over time, these classifications made it easier to track the performance of algorithms.

An influential line of research addressed the reproducibility of benchmarking studies by promoting common experimentation standards, including uniform initialization, consistent stopping criteria, and metrics (Moreau et al. 2022). Nevertheless, comprehensive standardization remained vague due to the wide range of algorithms, all of which interacted differently with the same function (Kistowski et al. 2015). An examination of published work points out that even if a universal testing framework seemed unattainable, researchers could still converge on shared best practices, such as reporting the number of function evaluations to reach particular thresholds and running experiments for multiple independent trials (López-Ibáñez et al. 2021; Sigmund 2022).

By the mid-2000s, specialized benchmarking platforms emerged, particularly in evolutionary computation competitions and blackbox optimization benchmarks (Martí et al. 2025). These initiatives provided official suites of test functions with known properties (van Thieu 2024). One notable outcome was that algorithm developers had to demonstrate competence in a community-recognized suite (Agushaka et al. 2022). The 2010–20s witnessed a surge of interest in benchmarking research, partly in response to concerns that differences in implementation details or hardware architectures confounded reported results (Skvorc et al. 2019). This sparked discussions around standardized libraries and code

repositories hosted on public platforms (Piotrowski et al. 2023; Mohamed et al. 2023; Mejía-de-Dios and Mezura-Montes 2022). Numerous papers showcased the advantages of distributing the algorithm code and reproducible workflow scripts (Kämpf et al. 2010; Hussain et al. 2017).

In most of the reviewed work above, test functions were seen to serve as proxy problems that help developers gauge how effectively an optimization algorithm converges, how stable its solutions are, and how well it scales (Brockhoff et al. 2015; Musafer et al. 2022; Venter 2010). Beyond performance metrics, developers can also examine runtime and memory consumption (Liang et al. 2005). A thorough benchmarking process can also reveal algorithmic properties such as premature convergence or a propensity to get trapped in local optima (Andrei 2008). In systematically analyzing performance on synthetic tests, one can better anticipate how an algorithm might behave on actual tasks and isolate factors that enhance or impede convergence by observing algorithmic behavior across various functions (Dieterich and Hartke 2012).

The role of test functions extends to establishing comparability among competing methods (Olson et al. 2017; Chen et al. 2022). Peer comparisons become possible when different research groups evaluate their algorithms on the same benchmark suite (Hoffmann et al. 2019). Moreover, test functions often serve pedagogical purposes, where students and practitioners new to optimization can use them to grasp tuning strategies or local search steps. Thus, test functions help bridge the gap between theory and application and gradually build an intuition for how algorithms respond to diverse problem characteristics (Kazemzadeh Azad and Kazemzadeh Azad 2023).

3 | Technical Definitions

We start with a catalog of technical definitions and details that will serve as a primer to this review. The readers are encouraged to review other sources for a more comprehensive discussion (Cavazzuti 2013; Diwekar 2021; Pedregal 2004; Adby 2013).

3.1 | Benchmark Function

A benchmark function is a well-defined mathematical expression constructed to evaluate the effectiveness of optimization algorithms under controlled conditions systematically. Such functions typically incorporate known properties such as the location and value of their global extrema and are designed to replicate the complexities encountered in practical optimization tasks. These complexities may include steep gradients, discontinuities, deceptive local minima, plateaus, and narrow basins of attraction.

3.2 | Continuity

A function f: $\mathbb{R}^n \to \mathbb{R}$ is said to be continuous at a point $x_0 \in \mathbb{R}^n$ *if for every* $\varepsilon > 0$ there exists a $\delta > 0$ such that $\|x - x_0\| < \delta$ implies $|f(x) - f(x_0)| < \varepsilon$. This definition reflects the intuitive notion of being able to trace the graph of f without sudden jumps or gaps (Terazono and Matani 2015). Continuity ensures that small perturbations in the input produce correspondingly small changes in the output. This property is essential in many optimization scenarios where gradient-based or local-search methods rely on smooth transitions to locate extrema.

3.3 | Convergence

Convergence describes the behavior of an iterative optimization algorithm as it progresses toward an optimal solution (Bull 2011). More precisely, an algorithm is said to converge to a point x^* if, for each $\varepsilon > 0$, there exists an iteration N such that for all $n \geq N$, the distance between the nth solution x_n and x^* is less than ε. Rapid convergence rates are typically desirable, but an overly aggressive approach can lead to premature convergence, where an algorithm settles on suboptimal local minima without adequately sampling the search space.

3.4 | Differentiability

A real-valued function f defined on $\mathbb{R}^n$ is differentiable at a point x_0 if there exists a linear map A: $\mathbb{R}^n \to \mathbb{R}$ such that $lim(x \to x_0)$ $(|f(x) - f(x_0) - A(x - x_0)|/\|x - x_0\|) = 0$. In one dimension, this condition reduces to the existence of a derivative, while in multiple dimensions, it involves partial derivatives (Sachs 1978). Differentiability is critical for methods that employ gradient-based information since derivatives indicate both the slope and direction of the steepest ascent (or descent) to enable more direct routes to local minima or maxima.

3.5 | Dimensionality

Dimensionality refers to the number of independent variables or degrees of freedom in a function or problem space. In an n-dimensional problem, each feasible solution can be represented as a point in $\mathbb{R}^n$, which grows exponentially in volume with increasing n. High-dimensional search spaces are characteristically difficult to navigate because many optimization techniques suffer from the "curse of dimensionality," wherein the computational or sampling effort increases considerably with dimension.

3.6 | Fitness Landscape

A fitness landscape is a conceptual topography in which each point in the search space is mapped to a real value representing its quality or fitness (Zou et al. 2022). Peaks, valleys, ridges, and plateaus in this space indicate high and low performance regions, guiding the trajectory of search heuristics. Visualizing or modeling the fitness landscape often provides insights into the complexity of the underlying function and the behavior of an algorithm attempting to discover globally optimal solutions.

3.7 | Global Optimum

A point x^* in the search space is the global optimum if $f(x^*) \leq f(x)$ for all x (in the case of minimization) or $f(x^*) \geq f(x)$ for all x (in

the case of maximization). Locating global optima is challenging in many real-world and theoretical contexts because landscapes often contain multiple basins of attraction and deceptive local optima. Algorithms seeking global solutions typically incorporate mechanisms (e.g., stochastic sampling or multi-start techniques) designed to avoid entrapment in non-global solutions.

3.8 | Local Optimum

A point x_0 is called a local optimum if there exists some neighborhood around x_0 within which x_0 yields the best function value. Formally, x_0 is a local minimum if there is a radius $\delta > 0$ such that $f(x_0) \leq f(x)$ for all x satisfying $\|x - x_0\| < \delta$. Unlike the global optimum, a local optimum is only superior to points in its immediate vicinity. The presence of multiple optima in high-dimensional or multimodal landscapes introduces additional complexity to search algorithms, which may need specialized mechanisms to escape locally optimal regions.

3.9 | Modality

Modality characterizes the number of distinct extrema (e.g., maxima or minima) a function possesses. A unimodal function has a single dominant optimum, whereas a multimodal function exhibits multiple optima. Higher modality typically increases the difficulty of an optimization task because algorithms must correctly navigate among several peaks or valleys to locate the global solution. Multimodal functions are thus valuable in benchmarking since they rigorously test an algorithm's resilience against premature convergence and its capacity to explore diverse regions of the search space.

3.10 | Noise and Stochasticity

Noise represents unpredictable or stochastic fluctuations that can corrupt the function's true evaluation, often stemming from measurement errors, uncertainties in model parameters, or numerical imprecision (Nissen and Propach 1998). For a function $f(x)$, the observed value may be $f(x) + \varepsilon$, where ε is a random variable. Stochasticity in the evaluation process can mislead deterministic search steps, necessitating robust optimization strategies to discern genuine improvements from random perturbations. In practice, noise resilience often involves statistical sampling, smoothing techniques, or adaptive parameter settings that tolerate fluctuating fitness scores.

3.11 | Optimization Algorithm

An optimization algorithm is a computational procedure devised to find the best solution to a given objective function within a specified domain. Such algorithms may leverage deterministic rules (e.g., gradient descent, Newton's method, and branch-and-bound) or stochastic heuristics (e.g., evolutionary strategies, simulated annealing, and swarm intelligence). Core components typically include initialization schemes, iterative update steps, and termination criteria. Effectiveness is measured by how well and quickly the algorithm locates globally optimal solutions while maintaining stability and scalability under various problem constraints.

3.12 | Scalability

Scalability indicates how consistently an algorithm or function retains its defining properties when extended to higher dimensions or larger problem instances (Maltese et al. 2018). For benchmark functions, scalability refers to preserving characteristic features (such as modality, separability, or smoothness) while dimension n is increased. For algorithms, it concerns whether performance degrades gracefully or dramatically as the problem size grows. Scalable test problems can reveal potential limitations of optimization methods in high-dimensional or expanding parameters.

3.13 | Search Space

The search space encompasses the entire set of candidate solutions over which an optimization algorithm operates. Formally, if the function domain is $D \subseteq \mathbb{R}^n$, then every $x \in D$ is a possible solution. The geometry and boundaries of this space, such as convexity or connectedness, can drastically influence algorithmic performance. Constrained problems restrict D through linear or nonlinear inequalities, adding complexity by limiting feasible regions and potentially partitioning the space into multiple disjoint subsets.

3.14 | Separability

A function $f(x_1, x_2, ..., x_n)$ is separable if it can be written as the sum of independent functions, each involving only one variable: $f(x_1, x_2, ..., x_n) = g_1(x_1) + g_2(x_2) + ... + g_n(x_n)$ (Hochbaum and Shanthikumar 1990). When this property holds, the original n-dimensional problem can be decomposed into n separate one-dimensional problems, greatly simplifying analysis and computational effort. Many optimization techniques exploit separability to reduce search complexity, enabling faster convergence and more direct gradient or direct search schemes.

4 | Classification of Benchmark Functions

This section provides a structured classification centered on three major dimensions: problem type, application domain, and complexity. Each dimension offers insights into some of the exemplary functions' defining features and the optimization challenges it represents.

4.1 | Function Type

Functions often differ substantially based on the nature of the search space, the number of objectives to optimize, and the presence or absence of constraints, among others. These characteristics shape the difficulty of the underlying optimization task and influence the relative performance of different algorithms (see Table 1).

TABLE 1 | Summary of function types.

Decision/question	Yes (→)	No (→)
Is the search space continuous?	Proceed to modality check	Select discrete functions (e.g., TSP and QAP)
Is the function unimodal?	Select unimodal continuous (Sphere, Rosenbrock, Elliptic)	Select multimodal continuous (Rastrigin, Ackley, Griewank)
Are variables separable?	Choose separable functions (Sphere, Step)	Choose non-separable functions (Rosenbrock variants)
Are constraints present?	Select constrained functions (penalty or repair)	Select unconstrained functions (full search space)

4.1.1 | Continuous Versus Discrete

Many benchmark functions in the optimization literature fall under the continuous category (Stein et al. 2004). Such functions present a search space defined over real-valued variables. Classical continuous benchmarks, such as those with smooth surfaces, highlight how gradient-based or gradient-free optimizers handle issues like local minima or narrow basins (Suwandi 2022). Continuous functions also allow for fine control over dimensionality and boundary constraints, which make them valuable for stress-testing gradient descent variants or evolutionary strategies. Examples of continuous functions include the Rosenbrock function, which has a narrow, curved valley that makes it difficult for simple algorithms to converge. The Rastrigin function incorporates multiple regularly spaced local minima, testing an algorithm's capability for global exploration. The Schwefel function is designed to lure solutions away from the global optimum by placing it on the domain boundary, highlighting the difficulty of balancing exploration against exploitation.

By contrast, discrete benchmarks restrict the domain to combinatorial configurations (Tilahun and Ngnotchouye 2017). Examples include variants of the traveling salesman problem (TSP) or integer-based knapsack formulations (Pop et al. 2024). Unlike continuous surfaces, discrete landscapes consist of discrete jumps between feasible solutions. This structural difference often renders gradient-based methods inapplicable and emphasizes the role of metaheuristics or specialized integer programming techniques. Discrete functions thus test an algorithm's ability to explore and exploit combinatorial neighborhoods efficiently, which makes them essential for evaluating population-based algorithms or specialized branch-and-bound strategies (Kaji 2021). For example, in TSP, the challenge is to find the minimal Hamiltonian cycle through a set of cities, typically defined by pairwise distances (Wang 2014). This problem exhibits a combinatorial increase in possible permutations, which can be used to measure an algorithm's performance regarding solution quality and computational effort. One example of discrete functions is the Quadratic Assignment Problem (QAP), which has combinatorial complexity (Loiola et al. 2007). QAP revolves around assigning a set of facilities to locations to minimize interaction costs. QAP's search space grows factorially with problem size, meaning metaheuristics or specialized integer programming methods ought to carefully balance intensification and diversification. These tasks do not rely on smooth gradient signals; instead, they push population-based or local search heuristics to discover near-optimal arrangements of discrete elements.

4.1.2 | Unimodal Versus Multimodal

Another possible way to categorize benchmark functions is by whether they are unimodal or multimodal. A unimodal function has a single global optimum, while a multimodal function features multiple locally optimal points, only one (or a few) of which is truly global (Droste et al. 1998). This distinction influences the difficulty of finding the best solution and highlights different optimization algorithms' contrasting strengths and weaknesses. A classic example of an unimodal function is the Sphere function, defined as:

$$f(x) = \sum_{i=1}^{d} x_i^2 \tag{1}$$

Since this function's contours are strictly convex, it offers no local minima apart from the single global minimum at $x=0$. The simplicity of such a function helps illustrate an algorithm's fundamental convergence properties, including how quickly it can reduce the objective value and how sensitive it might be to starting points or step sizes. Other functions with linear or quadratic bowls, like the Elliptic function, also fall into the unimodal category. The Rosenbrock function is also unimodal but more deceptive as its global minimum sits inside a slender, parabolic valley, making progress toward the optimum trickier than on the Sphere (Emiola and Adem 2021). Gradient-based methods often show slow convergence when the optimization path approaches the narrow valley walls.

As seen above, multimodal functions have two or more local optima, which challenges an algorithm's ability to conduct global exploration and escape from deceptive basins (Liu et al. 2011). The Rastrigin function exemplifies multimodality through a large number of regularly spaced local minima whose "wave-like" structure stems from embedded cosine terms. Thus, algorithms must avoid premature convergence in one of the many local minima. The Ackley function also exhibits multiple shallow local minima surrounding a deeper global optimum and is often used to test how well an algorithm navigates a relatively flat region before descending into a narrow basin (Wei et al. 2020).

Another notable multimodal benchmark is Griewank's function, whose product-based term induces multiple peaks and valleys throughout the domain.

Multimodal test functions examine algorithmic properties like diversification, exploration, and local search refinement in optimization algorithms. For instance, algorithms that rely primarily on local gradient information may converge to a suboptimal local solution unless they incorporate restarts, momentum, or adaptive step-size methods that enable them to break free of local traps. Meanwhile, population-based algorithms such as Genetic Algorithms, Particle Swarm Optimization, and other evolutionary methods can more effectively scan a broad search space if they maintain sufficient population diversity, which can help prevent premature convergence to a single local optimum.

4.1.3 | Separable Versus Non-Separable

In the context of optimization, a function is said to be separable if it can be expressed as a sum (or product, depending on the formulation) of individual sub-functions, each depending on a distinct variable (Stefanov 2021). Formally, a d-dimensional function is separable if it can be decomposed as.

$$f(x_1, x_2, \ldots, x_d) = \sum_{i=1}^{d} g_i(x_i) \tag{2}$$

for some set of single-variable functions $g_i(\cdot)$. Such a structure implies that each decision variable can be optimized independently of the others. Therefore, in benchmarking, separable functions are often used as baseline tests for algorithmic performance because they exhibit minimal interaction among variables. Optimization in these cases amounts to coordinating multiple independent one-dimensional searches. Algorithms that are adept at local search tend to perform well on separable benchmarks because improvements in one coordinate do not affect the others.

An important advantage of testing with separable functions is the opportunity to diagnose fundamental issues in an algorithm's search mechanism without the confounding effects of variable coupling. For instance, the aforementioned Sphere function is entirely separable and serves as a recognized test for convergence speed and numerical stability. Similarly, the Step function and some variants of Rosenbrock (when modified to remove cross-term coupling) can also be treated as nearly separable, which allows a straightforward analysis of how an algorithm handles simple unimodal or mildly multimodal landscapes.

Still, pure separability is relatively rare in real-world problems where design variables often interact, and hence, non-separable functions can more accurately reflect typical optimization challenges (Cárdenas-Montes et al. 2015). Therefore, a function is non-separable if it cannot be decomposed into lower-dimensional subproblems without losing critical coupling information. Mathematically, some components of the function depend on products or other interactions among the decision variables (Chandra et al. 2016). The Rosenbrock function, for example,

$$f(x_1, x_2) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2 \tag{3}$$

features cross-variable terms that make the global minimum path a curved valley rather than a simple alignment of independent minima. This cross-coupling tests an algorithm's ability to handle correlated parameters, where progress in one dimension necessitates compensatory changes in others.

4.1.4 | Differentiable Versus Non-Differentiable

Benchmark and test functions may also be categorized based on whether they are differentiable (Kiseleva and Stepanchuk 2003). Classical differentiable functions, such as the Sphere, Rosenbrock, and Ackley functions, often serve as testbeds for gradient-based methods because the algorithms rely on gradient information to guide the search (Daoud et al. 2023). In these contexts, differentiability allows for more precise steps: gradients inform local curvature, and higher-order methods can exploit Hessian approximations to accelerate convergence. Moreover, differentiable functions enable a variety of advanced theoretical analyses, such as local convergence rates and stability of critical points.

Non-differentiable functions can arise from absolute value terms, piecewise definitions, or discrete transitions (Zhou and Hu 2014). A classic example is the absolute value function in one dimension, $f(x) = |x|$, which is non-differentiable at $x = 0$. In higher dimensions, functions that incorporate *max* or *min* operations or model switching behavior in engineering systems exhibit corners or edges in their landscapes, thereby breaking differentiability. These discontinuities can occur in realistic situations, such as friction models in certain materials design tasks. These benchmark functions necessitate alternative optimization techniques that do not rely on smooth gradients, and hence, metaheuristics can be of interest as they are typically robust to non-smoothness and often explore the search space through population-based sampling (rather than gradient-based movements) (Cuevas et al. 2024). For example, functions like the Step or Schwefel's problem variants can contain flat plateaus and sharp discontinuities that highlight the limitations of gradient-based approaches while simultaneously testing the adaptability of metaheuristics. It is worth noting that in some cases, non-differentiability is introduced intentionally through constraints that translate into penalty functions. In such a scenario, if the penalty function is chosen as an indicator or step-based measure, it creates a non-differentiable region at the boundary between feasible and infeasible domains (Shor et al. 2012).

4.1.5 | Scalable Versus Non-Scalable

Scalability in benchmark functions refers to the capacity to adjust function size or dimensionality without qualitatively changing the problem's defining characteristics (Durillo, Nebro, Coello, et al. 2010). A scalable function is one that permits systematic exploration of an algorithm's performance as the problem dimension increases. For instance, the Sphere function mentioned above is trivially scalable, wherein a developer can add or remove dimensions while preserving its convex nature and quadratic contour. Similarly, functions like Ackley, Griewank, and Rastrigin have well-known d-dimensional generalizations, retaining key features such as multiple local minima or high ruggedness even when

dimension increases from 2 to hundreds or thousands of variables (Mahdavi et al. 2015).

By contrast, non-scalable functions are confined to fixed-size problems or lose their representative properties when scaled to higher dimensions (ECAI 2006). Some classical engineering benchmarks, such as certain truss design or gear train optimization problems, are defined with a rigid set of parameters and constraints tailored to a specific application (Öztürk and Kahraman 2023; Ben Younes et al. 2022). Attempting to scale up these benchmarks might entail introducing artificial variables or constraints that no longer reflect the original problem's physics/practical context. Similarly, discrete problems may not be trivially scalable in a meaningful way. While one can add more cities to the TSP, for instance, the resulting instance might differ substantially in complexity or fail to preserve the unique challenges of the original configuration. Therefore, balancing scalable and non-scalable benchmarks can foster a well-rounded analysis. Scalable functions reveal how an algorithm extends to large-scale or high-dimensional contexts, while non-scalable benchmarks offer fidelity to real complexities.

4.1.6 | Constrained Versus Unconstrained

Benchmark functions may include explicit constraints on the variable space to reflect physical, resource, or regulatory limitations found in real-world problems (Venkata Rao 2016; Lagaros et al. 2023). Constrained optimization requires algorithms to navigate feasible regions where constraints are satisfied, often via methods like penalty functions, feasibility repair heuristics, or multi-phase search strategies. These constraints can be linear or nonlinear, sharply limiting the permissible search domain or subtly guiding solutions along complex manifolds. Unconstrained functions omit such conditions and, by default, allow the entire search space to be explored (Nocedal 1992). While unconstrained benchmarks provide a foundational environment for testing an algorithm's raw optimization capability, they do not fully capture the complexity of real applications (Cui et al. 2017). Therefore, combining constraints with other challenging properties (like multimodality or high dimensionality) can help better evaluate whether an algorithm can maintain convergence quality when feasible regions become narrow or fragmented.

In addition to explicit constraints, real-world formulations often impose hidden constraints (i.e., limitations not directly expressed as equations or inequalities but emerge from practical considerations, such as computational budget or real-time responsiveness) (Hellwig and Beyer 2019). It is worth noting that such functions can be seen in certain Competitions on Evolutionary Computation (CEC) (Molina et al. 2018). Researchers/developers who seek to emulate these scenarios may add resource constraints, dynamic constraints, or time-sensitive constraints that push algorithms to find a feasible solution and one discoverable within strict operational windows (Gupta et al. 2021).

4.1.7 | Convex Versus Non-Convex

A further important distinction is the difference between convex and non-convex benchmarks (Yang 2010). Convex functions like the well-known Sphere function or simple quadratic bowls have a single global minimum with no local optima. Such functions offer a relatively direct path toward the optimum for gradient-based methods. In contrast, non-convex functions like Ackley, Rastrigin, or Griewank harbor multiple local minima or complex topographies that challenge any single-mode search method. These non-convex functions better represent the intricacies of real-world ML optimization tasks, where local traps, flat regions, and highly curved surfaces may be present.

4.1.8 | Single-Objective Versus Multi- and Many-Objectives

Single-objective functions, wherein a single metric of performance is optimized, remain the default choice for many foundational studies (Bonyadi and Michalewicz 2017). The simplicity of focusing on a single cost or fitness function facilitates clear comparisons of convergence speed, solution accuracy, and stability. Thus, developers can track whether an algorithm finds the global optimum, how many function evaluations are required, and how sensitive the solution is to initialization or parameter tuning. Some examples of single-objective functions can be seen in those listed under continuous functions.

On the other hand, multi- and many-objective benchmarks present two or more objectives that must be optimized simultaneously (Gunantara 2018). These objectives may conflict and hence require algorithms to produce solutions that trade off the objectives to varying extents, often captured by the concept of Pareto optimality. Popular multi-objective test suites featuring functions with known Pareto fronts allow developers to assess an algorithm's capability to uncover diverse non-dominated solutions. Such benchmarks challenge metaheuristics' exploration and exploitation mechanisms since balancing multiple objectives requires a careful blend of global search and local refinement. They also provide an opportunity to evaluate metrics like hypervolume or spread, revealing how effectively an algorithm approximates and distributes solutions along the true Pareto front. Some examples include the ZDT, DTLZ, and WFG families or gradient-based approaches that utilize Pareto-based selection (Yalaoui et al. 2021; Deb et al. 2006). ZDT1, for instance, provides a convex Pareto front that is relatively straightforward to capture, whereas ZDT4 introduces multimodality in the decision space. DTLZ functions allow control of problem dimensionality, the shape of the Pareto front, and the presence of local fronts. Each of these test suites evaluates a different aspect of multi-objective performance, from convergence speed to solution diversity.

4.2 | Application Domain

While function type offers a high-level perspective on how search spaces differ, application domain classifications reflect the contexts in which these functions are most relevant. The following sheds light on such use in ML, engineering, economics, and operations research. A similar discussion can be extended to other domains as well.

4.2.1 | Machine Learning

In ML settings, benchmark functions frequently mimic model loss landscapes or hyperparameter tuning objectives (Yang and Shami 2020). These functions range from simplified representations of typical error surfaces (e.g., quadratic bowls) to more intricate landscapes approximating neural network training scenarios (Agrawal 2020). Synthetic benchmarks in ML might introduce properties like non-convexity, high dimensionality, or noisy gradient estimates to emulate practical difficulties in training large models on real data. While exact modeling of such surfaces is computationally prohibitive, simplified versions exhibit features like plateau regions that can arise from widespread parameter symmetries or saddle points where gradient information is weak. For example, certain specially designed synthetic networks come with objective functions whose shape is analogous to that of a typical feedforward architecture. One might use similar functions to evaluate optimizer performance in avoiding saddle plateaus and dealing with stochastic gradient noise.

4.2.2 | Engineering, Economics, and Operations Research

Domains such as engineering design, resource allocation, or production scheduling require benchmark functions that incorporate feasibility conditions, multiple performance criteria, or dynamic constraints that shift over time. In engineering, for instance, benchmarks may simulate structural components under stress associated with constraints on material properties, weight, or cost (Gandomi and Yang 2011; Pereira et al. 2022). For example, a representative function might encode stress limit constraints as highly non-linear boundaries that must be respected while searching for the feasible region of minimum material usage. Another common example includes designing the cross-sectional area of a truss structure for minimal weight under stress constraints (Pereira et al. 2022).

Economic benchmarks can model supply–demand equilibria or investment portfolios balancing risk and return (Intriligator 2002). Operations research often involves discrete or mixed-integer benchmarks that test scheduling or routing decisions (David et al. 2015). For example, the flow shop scheduling problem can be formulated as an optimization function to minimize the makespan (the total completion time for a batch of jobs). The constraint is that each job must pass through machines in a specified order, resulting in combinatorial explosions as the problem size grows. A well-known variation is the permutation flow shop scheduling problem, which significantly tightens constraints by requiring all jobs to follow the same production sequence (Liang et al. 2022).

4.3 | Complexity

A third lens for classifying benchmark functions focuses on the complexity of the underlying problem. This dimension encompasses dimensionality, time-varying behavior, and computational overhead (see Figure 1).

4.3.1 | Low-Dimensionality

Low-dimensional benchmarks, typically in two or three dimensions, serve as illustrative examples for visual analysis and pedagogical understanding. Developers can plot contours or 3D surfaces to examine an algorithm's trajectory and tangibly analyze convergence phenomena. Despite their simplicity, these functions can still be challenging if they embed rugged regions or steep gradients. However, low-dimensional benchmarks often underrepresent the scale of modern optimization tasks.

4.3.2 | High-Dimensionality

As the number of dimensions grows, the search space expands exponentially. Population-based algorithms may struggle to maintain diversity, while gradient-based methods might face noisy or vanishing gradients. In ML scenarios, high-dimensional functions can emulate the objective surfaces associated with deep neural network training, which often feature numerous local optima and sharp ridges. Well-designed high-dimensional benchmarks incorporate not only increased variable counts but also realistic properties like correlated parameters or partial separability. For high-dimensional benchmarks, expansions of these same functions to n dimensions (such as the d-dimensional Rosenbrock or d-dimensional Rastrigin) demonstrate how quickly local optima and narrow valleys multiply in the search space. Meanwhile, the Elliptic function (a variant of the quadratic bowl) can become extremely difficult in high dimensions due to very different scaling factors along each principal axis, thereby requiring adaptive step sizes or advanced covariance matrix adaptation. The Hybrid Composition (HC) functions from certain competitions, like the CEC benchmark suites, combine multiple non-convex functions into a single, high-dimensional problem that better reflects complex real-world conditions by mixing different local surface structures (Mohamed et al. 2023).

4.3.3 | Dynamic or Time-Varying

Dynamic benchmarks modify the objective or constraint landscape over time to force algorithms to adapt (Xiang et al. 2022). These shifts may be periodic or unpredictable as a means to reflect real-world scenarios where data streams evolve or environmental conditions change (Yazdani, Nouhi et al. 2023). For example, a dynamic function might move its global optimum across the search space, alter constraint boundaries, or introduce new local optima. For instance, a given function may shift the global optimum smoothly along a predefined path, allowing an algorithm to anticipate and track the moving optima if it detects the shift pattern. Others may abruptly relocate optima, forcing a rapid re-exploration of the space. A well-known dynamic benchmark is the Moving Peaks Benchmark (MPB), which changes the location and height of several peaks in the search space as time advances (Moser and Chiong 2013; Fox et al. 2022). Algorithms suitable for dynamic optimization must balance the memory of prior solutions with continuous exploration to demonstrate agility in tracking moving targets.

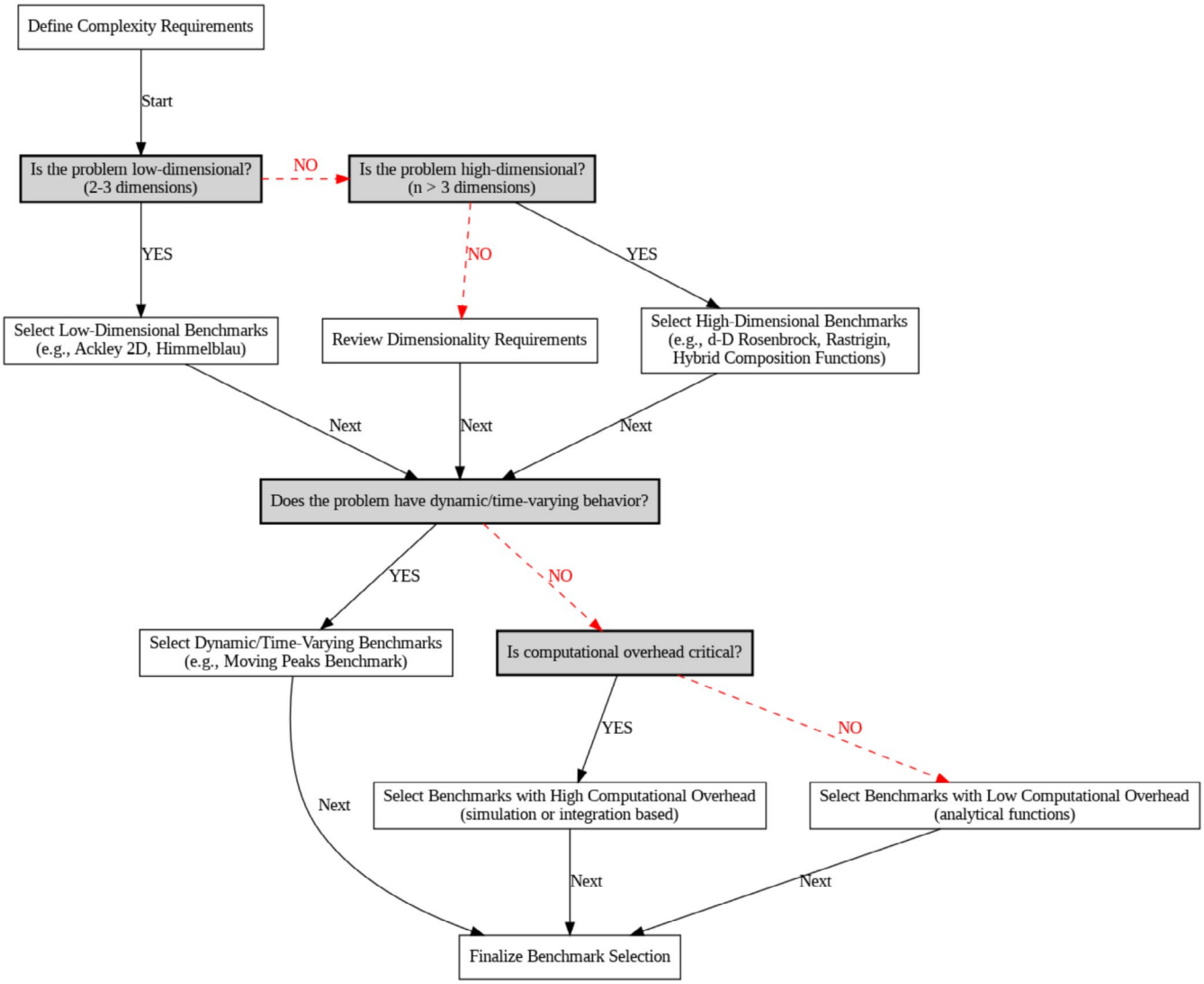

FIGURE 1 | Summary of complexity.

4.3.4 | Computational Overhead

In terms of computational overhead, certain benchmarks incorporate complex simulations or numerical integrations for function evaluations (Valiante et al. 2021). For instance, a single function evaluation in climate modeling may require running a partial differential equation solver or a detailed finite-element analysis, which can be prohibitively expensive. Metaheuristics or ML algorithms tested on such functions can emphasize computational efficiency, parallelization, or surrogate-based modeling. This computational facet is crucial for practical applicability since an algorithm with strong theoretical convergence but high evaluation demands may be unusable in time-sensitive engineering or ML tasks (Lan et al. 2022).

For completion, Table 2 further lists the functional properties of about 300 benchmark functions. A comprehensive review and information on each function is available in our recent review as well as in (Musafer et al. 2022; Dieterich and Hartke 2012; Kiseleva and Stepanchuk 2003; Molga and Smutnicki 2005; Jamil and Yang 2013; Ali et al. 2005; Rahnamayan et al. 2007; Test Functions Index, n.d.; Hansen et al. 1992). It is worth noting that the mathematical descriptions for these functions are spared herein but can be found in our earlier review (Naser et al. 2024).

4.4 | 25 of the Most Commonly Used Benchmark and Test Functions

Based on the above review and those found in (Brockhoff et al. 2015; Andrei 2008; Chandra et al. 2016; David et al. 2015; Liang et al. 2022; Xiang et al. 2022; Yazdani, Nouhi et al. 2023; Moser and Chiong 2013; Fox et al. 2022), we curate a list of the 25 most commonly used functions in optimizing ML models and metaheuristics. Here are these functions listed alphabetically (see Figure 2).

1. Ackley
2. Alpine
3. Beale's

TABLE 2 | List of examined functions.

No.	Function	Properties				
1	Ackley N.1	Continuous	Differentiable	Non-separable	Scalable	Multimodal
2	Ackley N.2	Continuous	Differentiable	Non-separable	Scalable	Unimodal
3	Ackley N.3	Continuous	Differentiable	Separable	Scalable	Unimodal
4	Ackley's Path	NA	NA	NA	NA	Multimodal
5	Ackley (Modified)	NA	NA	NA	NA	NA
6	Adjiman	Continuous	Differentiable	Separable	Non-scalable	Multimodal
7	Alpine N.1	Continuous	Differentiable	Separable	Non-scalable	Multimodal
8	Alpine N.2	Continuous	Differentiable	Separable	Scalable	Multimodal
9	Aluffi-Pentini	NA	NA	NA	NA	NA
10	Attractive Sector	NA	NA	NA	NA	Unimodal
11	Ais Parallel Hyper-Ellipsoid	NA	NA	NA	NA	Unimodal
12	Bartels Conn	Continuous	Non-differentiable	Separable	Scalable	Multimodal
13	Beale	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
14	Beale with Noise	NA	NA	NA	NA	NA
15	Becker Lago	NA	NA	NA	NA	Unimodal
16	Bent Cigar	NA	NA	Non-separable	NA	Unimodal
17	Bent Identity	NA	NA	NA	NA	NA
18	Biggs EXP2	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
19	Biggs EXP3	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
20	Biggs EXP4	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
21	Biggs EXP5	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
22	Bird	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
23	Bird with Noise	NA	NA	NA	NA	NA
24	Bohachevsky N.1	Continuous	Differentiable	Separable	Non-scalable	Multimodal
25	Bohachevsky N.2	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
26	Bohachevsky N.3	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal

(Continues)

TABLE 2 | (Continued)

No.	Function	Properties				
27	Booth	Continuous	Differentiable	Non-separable	Non-scalable	Unimodal
28	Boothby	NA	NA	NA	NA	Multimodal
29	Box Betts	NA	NA	NA	NA	Multimodal
30	Box Betts Quadratic Sum	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
31	Bradford	NA	NA	NA	NA	NA
32	Branin	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
33	Branin RCOS2	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
34	Brent	Continuous	Differentiable	Non-separable	Non-scalable	Unimodal
35	Brown	Continuous	Differentiable	Non-separable	Scalable	Unimodal
36	Brown Almost Linear	Continuous	Differentiable	Non-separable	Scalable	Unimodal
37	Bukin N.2	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
38	Bukin N.4	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
39	Bukin N.6	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
40	Camel Three Hump	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
41	Camel Six Hump	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
42	Carrom Table	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
43	Chameleon	NA	NA	NA	NA	NA
44	Chen Bird	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
45	Chen V	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
46	Chichinadze	Continuous	Differentiable	Separable	Non-scalable	Multimodal
47	Chung Reynolds	Continuous	Differentiable	Non-separable	Scalable	Unimodal
48	Chung Reynolds N.2	NA	NA	NA	NA	NA
49	Cigar	NA	NA	NA	NA	Multimodal
50	Clunar	NA	NA	NA	NA	NA
51	Cola	Continuous	Differentiable	Separable	Non-scalable	Multimodal
52	Colville	Continuous	Differentiable	Non-separable	Scalable	Multimodal

(Continues)

TABLE 2 | (Continued)

No.	Function	Properties				
53	Composite Griewank Rosenbrock F8F2	NA	NA	NA	NA	Multimodal
54	Composition Function N.1	NA	NA	NA	NA	NA
55	Composition Function N.2	NA	NA	NA	NA	NA
56	Corana	NA	NA	NA	Scalable	Multimodal
57	Cosine Envelope Sine Wave	NA	NA	NA	NA	NA
58	Cosine Function	NA	NA	NA	NA	NA
59	Cosine Mixture	Non-continuous	Non-differentiable	Non-separable	Non-scalable	Multimodal
60	Cosine Root	NA	NA	NA	NA	NA
61	Cosinee Mixture	NA	NA	NA	NA	NA
62	Cross In Tray	Continuous	Non-differentiable	Non-separable	Scalable	Multimodal
63	Cross Leg	NA	NA	NA	NA	NA
64	Cross Leg Table	NA	NA	NA	NA	Multimodal
65	Crowned Cross	NA	NA	NA	NA	Multimodal
66	Csendes	Continuous	Differentiable	Separable	Scalable	Multimodal
67	Cube	Continuous	Differentiable	Non-separable	Non-scalable	Unimodal
68	Damavandi	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
69	Damavandi N.2	NA	NA	NA	NA	NA
70	De Jong	Continuous	NA	NA	NA	Unimodal
71	De Jong N.5	NA	NA	NA	NA	Multimodal
72	Deb N.1	Continuous	Differentiable	Separable	Scalable	Multimodal
73	Deb N.3	Continuous	Differentiable	Separable	Scalable	Multimodal
74	Deb N.4	NA	NA	NA	NA	NA
75	Deckkers Aarts	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
76	Deflected Corrugated Spring	NA	NA	NA	NA	NA
77	Devillers Glasser	NA	NA	NA	NA	NA
78	Devilliers Glasser N.1	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal

(Continues)

TABLE 2 | (Continued)

No.	Function	Properties				
79	Devilliers Glasser N.2	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
80	Discus	NA	NA	NA	NA	Unimodal
81	Dixon Coles	NA	NA	NA	NA	NA
82	Dixon Price	Continuous	Differentiable	Non-separable	Scalable	Unimodal
83	Dixon Price N.2	NA	NA	NA	NA	NA
84	Dixon Price N.3	NA	NA	NA	NA	NA
85	Dixon Price N.4	NA	NA	NA	NA	NA
86	Dixon Price N.5	NA	NA	NA	NA	NA
87	Dolan	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
88	Drop Wave	NA	NA	NA	NA	Multimodal
89	Drop Wave N.2	NA	NA	NA	NA	NA
90	Dynamic Deceptive Basin	NA	NA	NA	NA	NA
91	Complex Dynamic Deceptive Basin	NA	NA	NA	NA	NA
92	Easom	Continuous	Differentiable	Separable	Scalable	Multimodal
93	Easom with Noise	NA	NA	NA	NA	NA
94	Egg Box	NA	NA	NA	NA	NA
95	Egg Crate	NA	NA	NA	NA	Unimodal
96	Egg Holder	Continuous	Differentiable	Non-separable	Scalable	Multimodal
97	Eggholder with Noise	NA	NA	NA	NA	NA
98	El Attar Vidyasagar Dutta	Continuous	Differentiable	Non-separable	Non-scalable	Unimodal
99	Elliptic	NA	NA	NA	NA	NA
100	Elliptic N.2	NA	NA	NA	NA	NA
101	Exp 2	NA	NA	NA	NA	NA
102	Exponential	Continuous	Differentiable	Non-separable	Scalable	Multimodal
103	Exponential Noise	NA	NA	NA	NA	NA
104	Fletcher Powell	NA	NA	NA	NA	NA

(Continues)

TABLE 2 | (Continued)

No.	Function	Properties				
105	Forrester	NA	NA	NA	NA	Multimodal
106	Fredrics Rcos	NA	NA	NA	NA	NA
107	Freudenstein Roth	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
108	Gaussian	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
109	Gaussian Perturbation	NA	NA	NA	NA	NA
110	Gear	NA	NA	NA	NA	NA
111	Gear N.2	NA	NA	NA	NA	NA
112	Giunta	Continuous	Differentiable	Separable	Scalable	Multimodal
113	Goldstein Price	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
114	Goldstein Price with Noise	NA	NA	NA	NA	NA
115	Gramacy Lee	NA	NA	NA	NA	NA
116	Griewank	NA	NA	NA	NA	NA
117	Griewank with Noise	NA	NA	NA	NA	NA
118	Gulf Research	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
119	Gulf Research and Development	NA	NA	NA	NA	NA
120	Hansen	Continuous	Differentiable	Separable	Non-scalable	Multimodal
121	Happy Cat	NA	NA	NA	NA	Multimodal
122	Hartman 3	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
123	Hartmann 3D	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
124	Helical Valley	Continuous	Differentiable	Non-separable	Scalable	Multimodal
125	Himmelblau	Continuous	Differentiable	Separable	Non-scalable	Multimodal
126	Holder Table 1	Continuous	Differentiable	Separable	Non-scalable	Multimodal
127	Holder Table 2	Continuous	Differentiable	Separable	Non-scalable	Multimodal
128	Holder Table N.2	NA	NA	NA	NA	NA
129	Holzman	NA	NA	NA	NA	NA
130	Hosaki	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal

(Continues)

TABLE 2 | (Continued)

No.	Function	Properties				
131	Hosaki Exponential	NA	NA	NA	NA	NA
132	Hosaki Exponential with Noise	NA	NA	NA	NA	NA
133	Hqing	NA	NA	NA	NA	NA
134	Hump	NA	NA	NA	NA	NA
135	Infinity	NA	NA	NA	NA	NA
136	Jennrich Sampson	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
137	Judge	NA	NA	NA	NA	NA
138	Katsuura	NA	NA	NA	NA	NA
139	Keane	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
140	Keane N.2	NA	NA	NA	NA	NA
141	Kowalik	NA	NA	NA	NA	NA
142	Langermann	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
143	Langermann N.2	NA	NA	NA	NA	NA
144	Lennard Jones	NA	NA	NA	NA	NA
145	Lennard Jones Minimum	NA	NA	NA	NA	NA
146	Leon	NA	NA	NA	NA	NA
147	Levi	NA	NA	NA	NA	NA
148	Levy Function	NA	NA	NA	NA	NA
149	Levy N.5	NA	NA	NA	NA	NA
150	Levy N.8	NA	NA	NA	NA	NA
151	Levy N.9	NA	NA	NA	NA	NA
152	Levy N.13	NA	NA	NA	NA	NA
153	Ljeunard Jones Minimum Boundary	NA	NA	NA	NA	NA
154	Matyas	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
155	McCormick	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
156	McCormick with Noise	NA	NA	NA	NA	NA

(Continues)

TABLE 2 | (Continued)

No.	Function	Properties				
157	Michalewicz	NA	NA	NA	NA	Multimodal
158	Michalewicz N.2	NA	NA	NA	NA	NA
159	Michalewicz with Noise	NA	NA	NA	NA	NA
160	Miele Cantrell	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
161	Mishra	Continuous	Differentiable	Non-separable	Scalable	Multimodal
162	Mishra N.2	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
163	Mishra N.3	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
164	Mishra N.4	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
165	Mishra N.5	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
166	Mishra N.6	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
167	Mishra N.7	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
168	Mishra N.8	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
169	Mishra N.9	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
170	Mishra N.10	Continuous	Differentiable	Non-separable	Scalable	Multimodal
171	Mishra N.11	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
172	Mishra's Bird	Continuous	Differentiable	Non-separable	Scalable	Multimodal
173	Muller Brown	NA	NA	NA	NA	NA
174	Nesterov Smooth Chebyshev Rosenbrock	NA	NA	NA	NA	NA
175	Parsopoulos	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
176	Pathological	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
177	Paviani	Continuous	Differentiable	Non-separable	Scalable	Multimodal
178	Pen Holder	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
179	Penalty	Continuous	Differentiable	Non-separable	NA	NA
180	Penalty N.2	Continuous	Differentiable	Non-separable	NA	NA
181	Perez Reche	NA	NA	NA	NA	NA
182	Periodic	Continuous	Differentiable	Non-separable	Scalable	Multimodal

(Continues)

TABLE 2 | (Continued)

No.	Function	Properties				
183	Perm	NA	NA	NA	NA	Unimodal
184	Perm Function 0,d beta	NA	NA	NA	NA	Unimodal
185	Pinter	Continuous	Differentiable	Non-separable	Scalable	Multimodal
186	Plateau	NA	NA	NA	NA	NA
187	Powell	Continuous	Differentiable	Non-separable	Scalable	Unimodal
188	Powell Singular (aka. Powell's Quartic Function)	Continuous	Differentiable	Non-separable	Scalable	Unimodal
189	Powell Singular N.2	NA	NA	NA	NA	NA
190	Powell Sum	Continuous	Non-differentiable	Separable	Scalable	Multimodal
191	Power Sum	NA	NA	NA	NA	Unimodal
192	Price N.1	Continuous	Non-differentiable	Separable	Non-scalable	Multimodal
193	Price N.2	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
194	Price N.3	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
195	Price N.4	Continuous	Differentiable	Separable	Scalable	Multimodal
196	Prices	NA	NA	NA	NA	NA
197	Qing	Continuous	Differentiable	Separable	Scalable	Multimodal
198	Qing Variant	NA	NA	NA	NA	NA
199	Qing N.2	NA	NA	NA	NA	NA
200	Quadratic	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
201	Quartic	Continuous	Differentiable	Separable	Scalable	NA
202	Quintic	Continuous	Differentiable	Separable	Non-scalable	Multimodal
203	Rana	Continuous	Differentiable	Non-separable	Scalable	Multimodal
204	Rastrigin	NA	NA	Non-separable	NA	Multimodal
205	Rastrigin Modified	NA	NA	NA	NA	Multimodal
206	Rayleigh	NA	NA	NA	NA	NA
207	Ridge	NA	NA	NA	NA	NA
208	Ripple	NA	NA	Non-separable	NA	NA

(Continues)

TABLE 2 | (Continued)

No.	Function	Properties				
209	Ripple N.25	NA	NA	Non-separable	NA	NA
210	Rosenbrock	Continuous	Differentiable	Non-separable	Scalable	Unimodal
211	Rosenbrock Modified	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
212	Rotated Ellipse	Continuous	Differentiable	Non-separable	Non-scalable	Unimodal
213	Rotated Ellipse N.2	Continuous	Differentiable	Non-separable	Non-scalable	Unimodal
214	Rotated Hyper Ellipsoid	NA	NA	NA	NA	NA
215	Rump	Continuous	Differentiable	Non-separable	Non-scalable	Unimodal
216	Salomon	Continuous	Differentiable	Non-separable	Scalable	Multimodal
217	Sargan	Continuous	Differentiable	Non-separable	Scalable	Multimodal
218	Sawtooth	NA	NA	NA	NA	NA
219	Schaffer N.1	Continuous	Differentiable	Non-separable	Non-scalable	Unimodal
220	Schaffer N.2	Continuous	Differentiable	Non-separable	Non-scalable	Unimodal
221	Schaffer N.3	Continuous	Differentiable	Non-separable	Non-scalable	Unimodal
222	Schaffer N.4	Continuous	Differentiable	Non-separable	Non-scalable	Unimodal
223	Schaffer F6	Continuous	Differentiable	Non-separable	Scalable	Multimodal
224	Schmidt Vetters	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
225	Schumer Steiglitz	NA	NA	NA	NA	NA
226	Schwefel	Continuous	Differentiable	Non-separable	Scalable	Unimodal
227	Schwefel 1.2	Continuous	Differentiable	Non-separable	Scalable	Multimodal
228	Schwefel 2.20	Continuous	Non-differentiable	Separable	Scalable	Unimodal
229	Schwefel 2.21	Continuous	NA	NA	NA	Unimodal
230	Schwefel 2.22	Continuous	NA	Separable	NA	Unimodal
231	Schwefel 2.23	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
232	Schwefel 2.23A	NA	NA	NA	NA	NA
233	Schwefel 2.25	Continuous	Differentiable	Separable	Non-scalable	Multimodal

(Continues)

TABLE 2 | (Continued)

No.	Function	Properties				
234	Schwefel 2.26	Continuous	Differentiable	Separable	Non-scalable	Multimodal
235	Schwefel 2.36	Continuous	Differentiable	Separable	Scalable	Multimodal
236	Schwefel 2.40	Continuous	Differentiable	Separable	Non-scalable	Multimodal
237	Schwefel 2.60	Continuous	Non-differentiable	Separable	Scalable	Unimodal
238	Shekel N.10	Continuous	Differentiable	Non-separable	Scalable	Multimodal
239	Shekel N.5	Continuous	Differentiable	Non-separable	Scalable	Multimodal
240	Shekel N.7	Continuous	Differentiable	Non-separable	Scalable	Multimodal
241	Shubert	Continuous	Differentiable	Separable	Non-scalable	Multimodal
242	Shubert N.3	Continuous	Differentiable	Separable	Non-scalable	Multimodal
243	Shubert N.4	Continuous	Differentiable	Separable	Non-scalable	Multimodal
244	Sine Cosine Algorithm	NA	NA	NA	NA	NA
245	Sine Envelope Sine Wave	NA	NA	NA	NA	NA
246	SODP	NA	NA	NA	NA	NA
247	Sphere	Continuous	Differentiable	Separable	Scalable	Multimodal
248	Step N.2	Non-continuous	Non-differentiable	Separable	Scalable	Unimodal
249	Step N.3	Non-continuous	Non-differentiable	Separable	Scalable	Unimodal
250	Step 4,7	Non-continuous	Non-differentiable	Separable	Scalable	Multimodal
251	Stepint	Non-continuous	Non-differentiable	Separable	Scalable	Unimodal
252	Stochastic	NA	NA	NA	NA	NA
253	Stretched Cosine Wave	NA	NA	NA	NA	NA
254	Stretched V Sine Wave	Continuous	Differentiable	Non-separable	Scalable	Unimodal
255	Styblinski Tang	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
256	Sum of Different Powers	NA	NA	NA	NA	Unimodal
257	Sum Squares	Continuous	NA	NA	NA	Unimodal
258	Tablet	NA	NA	NA	NA	NA

(Continues)

TABLE 2 | (Continued)

No.	Function	Properties				
259	Testtube Holder	Continuous	Differentiable	Separable	Scalable	Multimodal
260	Thurber	NA	NA	NA	NA	NA
261	Trec	NA	NA	NA	NA	NA
262	Trecanni	Continuous	Differentiable	Separable	Scalable	Unimodal
263	Trefethen	NA	NA	NA	NA	Multimodal
264	Trid	NA	NA	NA	NA	Unimodal
265	Trid N.6	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
266	Trid N.8	NA	NA	NA	NA	NA
267	Trid N.10	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
268	Trigonometric	Continuous	Differentiable	Non-separable	Scalable	Unimodal
269	Trigonometric N.2	Continuous	Differentiable	Non-separable	Scalable	Multimodal
270	Trimmed Sphere	Continuous	Non-differentiable	NA	NA	NA
271	Tripod	NA	NA	Non-separable	NA	Multimodal
272	Two Axis	NA	NA	NA	NA	NA
273	Urfun N.2	NA	NA	NA	NA	NA
274	Ursem	NA	NA	NA	Non-scalable	NA
275	Ursem N.3	NA	NA	NA	Non-scalable	NA
276	Ursem N.4	NA	NA	NA	Non-scalable	Unimodal
277	Ursem Waves	NA	NA	NA	NA	NA
278	Ursem Wavesfun N.2	NA	NA	NA	Non-scalable	Unimodal
279	Venter Sobiezcczanski Sobieski	Continuous	Differentiable	Separable	Non-scalable	NA
280	Vincent	NA	NA	NA	NA	NA
281	Watson	Continuous	Differentiable	Non-separable	Scalable	Unimodal
282	Wavy	Continuous	Differentiable	Separable	Scalable	Multimodal
283	Wayburn Seader	NA	NA	NA	NA	NA

(Continues)

TABLE 2 | (Continued)

No.	Function	Properties				
284	Wayburn Seader N.2	Continuous	Differentiable	Non-separable	Scalable	Unimodal
285	Weierstrass	Continuous	Differentiable	NA	Scalable	Multimodal
286	Whitley	Continuous	Differentiable	Non-separable	Scalable	Multimodal
287	Wolfe	Continuous	Differentiable	Separable	Scalable	Multimodal
288	Xin She Yang N.1	NA	NA	Non-separable	NA	NA
289	Xin She Yang N.2	NA	NA	Separable	NA	NA
290	Xin She Yang N.3	NA	NA	Non-separable	NA	NA
291	Xin She Yang N.4	NA	NA	NA	Non-scalable	NA
292	Xin She Yang N.7	NA	NA	NA	NA	NA
293	Yao Liu N.4	NA	NA	NA	Non-scalable	NA
294	Yao Liu N.9	NA	NA	NA	NA	NA
295	Zakharov	Continuous	Differentiable	Non-separable	Scalable	Multimodal
296	Expanded Zakharov	NA	NA	NA	NA	NA
297	Zero Sum	NA	NA	NA	NA	NA
298	Zettl	Continuous	Differentiable	Non-separable	Non-scalable	Multimodal
299	Zettl Variant	NA	NA	NA	NA	NA
300	Zoom	NA	NA	NA	NA	NA

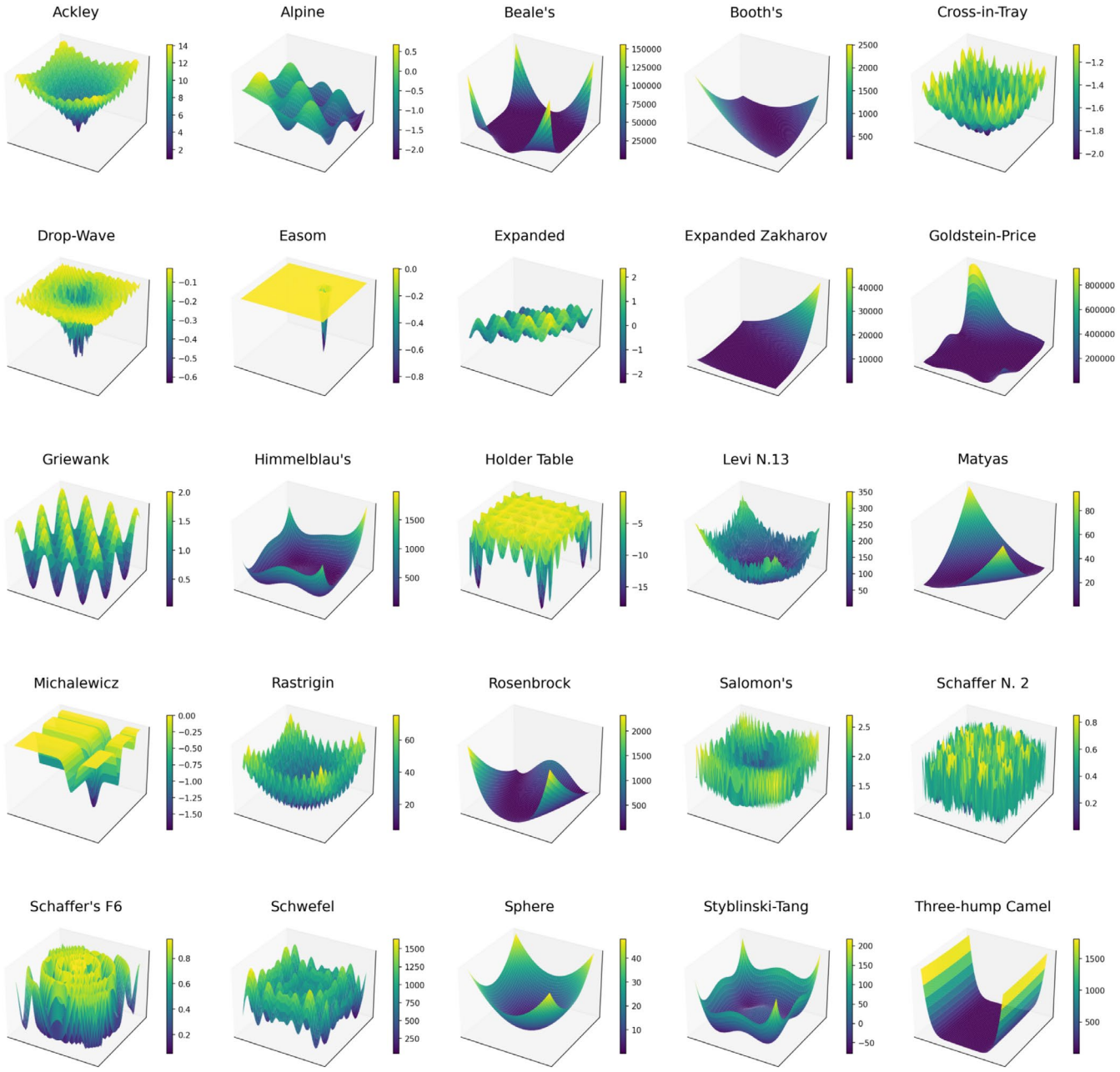

FIGURE 2 | 25 most commonly used functions.

4. Booth's
5. Cross-in-Tray
6. Drop-Wave
7. Easom
8. Expanded
9. Schaffer's F6
10. Expanded Zakharov
11. Goldstein-Price
12. Griewank
13. Himmelblau's
14. Holder Table
15. Levi N.13
16. Matyas
17. Michalewicz
18. Rastrigin
19. Rosenbrock
20. Salomon's
21. Schaffer N. 2
22. Schwefel
23. Sphere

24. Styblinski-Tang
25. Three-hump Camel

4.5 | New Functions

As seen above, there is interest in dynamic optimization problems where the landscape changes over time. In order to bridge this gap, we present two new functions (see Figure 3). These are called,

- The *Dynamic Deceptive Basin* function: This function is dynamic, meaning its landscape changes over time as it incorporates a stochastic element through a random walk in the parameter space. This dynamic nature requires algorithms to adapt continuously to find and maintain optimal solutions, which is a valuable trait for a benchmark function. The addition of noise further complicates the optimization process, testing the algorithm's ability to differentiate between true gradient changes and random fluctuations.

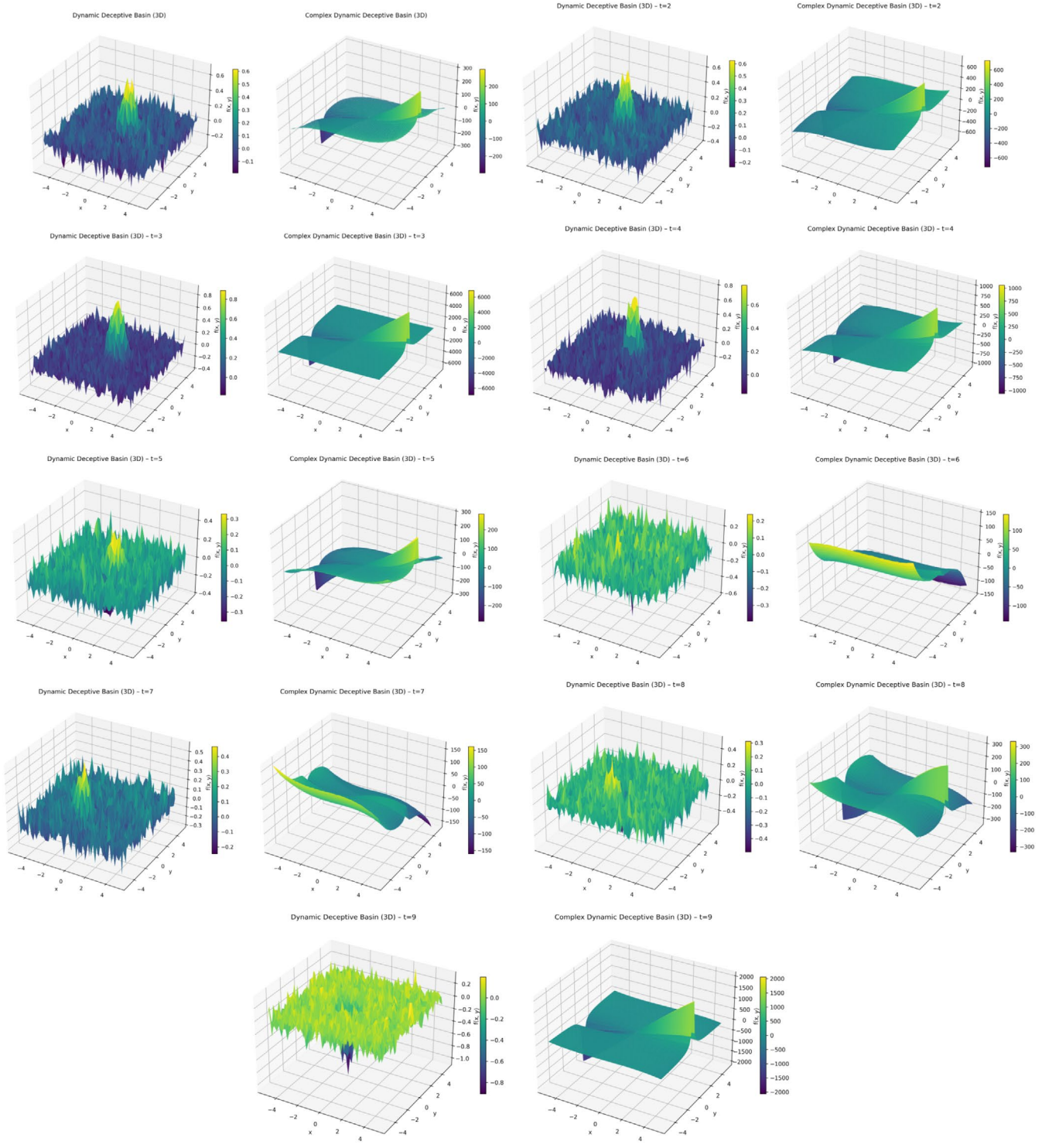

FIGURE 3 | Landscape of the newly proposed functions ($t=0$ to $t=9$).

$$f(x) = \sin\left(x_0 + \theta_0^{(t)}\right) \cdot \cos\left(x_1 + \theta_1^{(t)}\right) \cdot e^{-\|x-\theta^t\|^2} + \epsilon_x \quad (4)$$

where: $x = [x_0, x_1, ..., x_{n-1}]$ represents the input vector. $\theta^t = [\theta_0^{(t)}, \theta_1^{(t)}, ..., \theta_{n-1}^{(t)}]$ represents the dynamically updating state parameters. Note that $\theta_i^{(t)}$ is the i-th component (dimension) at time t. ϵ_x is normally noise distributed random variable ($\sim(0, 0.1)$).

- The *Complex Dynamic Deceptive Basin* function: This function is considerably more complex than its counterpart. It integrates polynomial, trigonometric, interaction terms, and exponential components, each adding to the function's multimodality and non-linearity. The historical influence on the function's behavior adds a layer of complexity, simulating real-world scenarios where past states affect current conditions.

$$f(x) = \sum_{i=0}^{n-1} \theta_0^{(t)} x_i^3 + \sum_{i=0}^{n-1} \sin\left(x_i + \theta_1^{(t)}\right) + \sum_{i=0}^{n-1} \frac{x_i \left(\theta_i^{(t)}\right)^{i+n}}{0.1\left|x_2 \theta_n^{(t)}\right|} + \sum_{i=0}^{n-1} \left(\theta_i^{(t)}\right)^2 + \epsilon_x \quad (5)$$

where: $x = [x_0, x_1, ..., x_{n-1}]$ represents the input vector. $\theta^t = [\theta_0^{(t)}, \theta_1^{(t)}, ..., \theta_{n-1}^{(t)}]$ represents the dynamically updating state parameters. Note that $\theta_i^{(t)}$ is the i-th component (dimension) at time t. ϵ_x is noise with a variance dependent on the norm of $-\sin$(recent effect) and is normally distributed, $\epsilon_x \sim (0,$ noise amplitude).

As one can see, both of these functions are non-stationary and contain multiple types of mathematical operations to create numerous local minima and maxima (i.e., intricate interdependencies). They also contain noise caused by random perturbations. The functions incorporate random noise and dynamic parameters that change with each function evaluation. This means that the landscape of the function is constantly shifting, making it harder for optimization algorithms to converge to a stable solution. It is worth noting that the *Complex Dynamic Deceptive Basin* function incorporates a history of previous solutions, effectively introducing memory effects into the optimization process.

5 | Evaluation Metrics and Methodologies

Evaluation metrics and experimental methods form the backbone of assessment in optimization and dictate how algorithmic performance is measured, compared, and reported (Riquelme et al. 2015; Naser and Amir 2021). In the context of benchmark and test functions, an appropriate selection of metrics ensures that we can systematically assess how quickly and reliably algorithms converge (Tian et al. 2024). Moreover, well-designed experimental protocols, including clear statistical validation and visualization strategies, facilitate reproducibility and fair comparisons across different studies (Huang et al. 2021). This section explores these aspects by covering performance metrics, statistical methods, and visualization techniques predominantly used in optimization research.

5.1 | Performance Metrics

A core issue in evaluating optimization methods is deciding which metrics to prioritize (Mirjalili and Lewis 2015). Although convergence speed and final solution quality are prominent, additional factors such as computational cost, robustness to noise, and scalability across problem sizes often distinguish methods that excel in practice from those in controlled experiments (ECAI 2006). The emphasis on any of these elements depends heavily on the application domain, the test function's complexity, and the constraints intrinsic to the problem.

Convergence speed constitutes one of the most frequently cited measures (Durillo, Nebro, Luna, et al. 2010). Developers typically track the number of function evaluations or iterations required to reach a predefined accuracy threshold, such as a small tolerance away from the known global optimum or a limit based on acceptable performance within a practical context (Kazikova et al. 2021). Algorithms that minimize an objective in fewer evaluations demonstrate more efficient exploration–exploitation dynamics (Nebro et al. 2008). Developers typically track one of the following:

- Number of function evaluations

Let $f(x)$ be the objective function, with $x \in \mathbb{R}^n$. Suppose the known global optimum is $f(x^*)$. A straightforward metric is the total number of function evaluations *Neval* needed to meet a performance criterion, such as.

$$|f(x_k) - f(x^*)| \leq \epsilon \quad (6)$$

where $\epsilon > 0$ is a chosen tolerance. Algorithms requiring fewer evaluations for the same ϵ are generally considered to have faster convergence.

- Convergence rate

The convergence rate is often classified as sublinear, linear, or superlinear, according to how fast the error $e_k = |x_k - x^*|$ or the objective gap $\left|f(x_k) - f(x^*)\right|$ decays as k (the iteration counter) increases.

- Linear convergence

We say $\{x_k\}$ converges *linearly* to x^* if there exist constants C and k_0 with $0 < C < 1$, $k_0 \geq 0$, such that for all $k \geq k_0$,

$$|x_{k+1} - x^*| \leq C\,|x_k - x^*| \quad (7)$$

Equivalently, $limsup_{k\to\infty} \frac{|x_{k+1}-x^*|}{|x_k-x^*|} = C < 1$, which implies the error decreases at a geometric (constant-factor < 1) rate.

- Sublinear convergence

If $|x_k - x^*| \to 0$ but $limsup_{k\to\infty} \frac{|x_{k+1}-x^*|}{|x_k-x^*|} = 1$,the rate is called *sublinear* (e.g., $(1/k)$ *or* $O(1/k^2)$).

- Superlinear convergence

If $limsup_{k\to\infty} \frac{|x_{k+1}-x^*|}{|x_k-x^*|} = 0$, the method is *superlinear* (e.g., quadratic convergence).

However, focusing solely on convergence speed can be misleading if it comes at the expense of generalizability or robustness. For example, an optimizer might rapidly converge on smooth,

low-dimensional problems but fail to scale to larger, multimodal, or highly constrained domains.

Accuracy or solution quality assesses how closely an algorithm's final solution approaches the global optimum (or a set of Pareto-optimal solutions in multi-objective settings) (Rahkar Farshi 2021).

- **Single-objective benchmarks** typically define a numeric global optimum; measuring the difference between the algorithm's best outcome and that optimum provides a direct indication of performance. For example,

If $f(x^*)$is known, one measure (absolute error),

$$\Delta_{\text{absolute error}} = f\left(x_{best}\right) - f(x^*) \tag{8}$$

And another measure (relative error),

$$\Delta_{\text{relative error}} = \frac{f\left(x_{best}\right) - f(x^*)}{f(x^*)} \tag{9}$$

where x_{best} is the final or best-found solution. A smaller Δ_{final} indicates better accuracy. These measures can be used simultaneously.

- **Multi-objective indicators**, like hypervolume: for a set of solutions $S \subset \mathbb{R}^n$ approximating the Pareto front, one common measure is the hypervolume of the dominated portion of the objective space:

$$HV(S) = \lambda\left(\left[r, f(s)\right]\right) \tag{10}$$

where r is a reference point, $f(s)$ maps a solution s to the objective space, and $\lambda(\cdot)$ denotes the Lebesgue measure (e.g., volume) of the union of all dominated hyper-rectangles. A larger hypervolume implies a more comprehensive coverage of the Pareto front. It is worth noting that coverage-based metrics count how many Pareto-optimal points in one set dominate or are dominated by another set, thus comparing two solution fronts.

Constraint handling

When constraints $g_i(x) \leq 0$ and $h_j(x) = 0$ are present, one often uses a penalty approach or discards solutions violating the constraints (Rahimi et al. 2023). For example, solutions may be ranked based on.

$$f_{penalized}(x) = f(x) + \sum_i \alpha_i \, max\left(0, g_i(x)\right) + \sum_j \beta_i \left|h_j(x)\right| \tag{11}$$

where α_i and β_j are penalty coefficients. Final accuracy metrics then consider this penalized objective function.

Computational cost encompasses not only raw execution time but also memory usage and any overhead associated with the algorithm's internal components. More specifically,

- Runtime

The elapsed time or wall-clock time for the entire optimization run (Neumann and Witt 2009). In algorithmic complexity terms, one sometimes expresses cost as $O(T(n,d))$, where n denotes the dimensionality and d might denote any additional parameters (e.g., population size in evolutionary algorithms).

- Memory usage

Large-scale methods often maintain multiple candidate solutions (e.g., populations in genetic algorithms) or store complex gradient information (e.g., in quasi-Newton approaches) (Anil et al. 2019). A memory overhead metric *Musage* can track the maximum or cumulative storage requirements.

When function evaluations themselves are expensive, the overall computational cost can be dominated by $N_{eval} \times C_{eval}$, where the latter is the cost of a single evaluation. Therefore, researchers often balance convergence speed against resource consumption as some improvements in solution quality may not justify growth in computational demands.

Robustness to noise is another key consideration since many real-world tasks and advanced synthetic benchmarks incorporate uncertainty or stochastic components to reflect measurement errors or dynamic changes (Deb and Gupta 2006). Hence, researchers frequently measure robustness by conducting multiple runs under varied noise seeds or distribution parameters and then examining average solution quality, standard deviations, or worst-case outcomes (Beyer and Sendhoff 2007). For example:

- Statistical robustness

This can be assessed by running the algorithm multiple times with different random seeds or noise realizations:

$$\left\{x_{best}^{(r)}, f\left(x_{best}^{(r)}\right)\right\}_{r=1}^{R} \tag{12}$$

then analyzing mean performance, standard deviations, or worst-case outcomes $max_1 \leq r \leq R \, f(x_{best}{}^{(r)})$ for minimization. An algorithm is considered robust if it yields consistently good solutions with low variance.

- Noise-resilient search strategies

Techniques such as repeated sampling at candidate points, smoothing approaches, or Bayesian methods that explicitly model the noise distribution can mitigate the effects of high-variance function evaluations. However, these additions can also increase computational costs. Steep deterioration under slight increases in noise signals a potential vulnerability in the search strategy.

Scalability tests how an algorithm's performance evolves as dimensionality or problem size increases (Maltese et al. 2018; Steenkamp and Engelbrecht 2021). Scalability can be important when examining techniques that display impressive results on smaller benchmarks but may degrade significantly when confronted with hundreds or thousands of decision variables. Scalability can be quantified by:

- Growth of function evaluations or runtime

One tracks how N_{eval} or the total runtime T_{exec} scales with n. For example, a polynomial relationship $T_{exec} = O(n^2)$ is more scalable than an exponential one $O(2^n)$.

- Trade-off between accuracy and dimensionality

Holding the iteration budget or total runtime fixed, one can observe the change in accuracy (e.g., Δ_{final}) as n increases. A method is considered to have good scalability if it can maintain acceptable solution quality for higher-dimensional problems without incurring prohibitive computational costs.

5.2 | Statistical Validation

As noted above, random initializations, stochastic operators, or noise in the function evaluation can introduce variability into performance metrics (Abdel-Basset et al. 2023). Hence, replicating experiments with multiple independent runs becomes essential to examine the performance of algorithms across trials. Thus, once performance metrics are collected, a robust statistical analysis can be conducted to ensure that the observed algorithm differences are not attributable to random chance (Bartz-Beielstein et al. 2010; ECAI 2006).

A variety of statistical tests can be carried out. For example, statistical tests such as the Wilcoxon signed-rank test, Friedman test, or analysis of variance (ANOVA) are commonly employed to compare algorithmic performance (Farag et al. 2020; Xie et al. 2021; Dobslaw 2010). In more detail, the Wilcoxon signed-rank test operates as a non-parametric alternative to the paired t-test, which is particularly valuable when distributional assumptions cannot be verified (Krishnamoorthy 2020). This test's mechanism involves ranking the absolute differences between paired observations and analyzing the number of ranks where the difference is positive. This test demonstrates robust statistical power when evaluating pairwise algorithmic comparisons across benchmark instances, especially with smaller sample sizes where normality assumptions may be violated. Simply, the test's statistical significance directly indicates whether one algorithm consistently outperforms another across the test instances.

On the other hand, the Friedman test extends non-parametric analysis to multiple algorithm comparison scenarios, functioning essentially as a non-parametric counterpart to repeated-measures ANOVA (Sheldon et al. 1996). This test operates by ranking each algorithm's performance separately within each benchmark instance and then analyzing the distribution of these ranks across algorithms. This methodology effectively controls for instance-specific variability by treating each benchmark as a blocking factor. The resulting statistic approximation allows developers to determine whether performance differences exist among the algorithmic set as a collective.

Analysis of variance (ANOVA) represents a parametric approach requiring assumptions of normality and homoscedasticity (Sthle and Wold 1989). While potentially more effective when its assumptions are satisfied, ANOVA's application demands careful validation of these statistical prerequisites. The F-statistic computation partitions total variance into between-algorithm and within-algorithm components to enable precise quantification of the relationship between algorithmic choice and performance variability.

In general, the Wilcoxon test often appears in pairwise comparisons, especially for smaller sample sizes, whereas ANOVA or Friedman can compare multiple algorithms simultaneously. Follow-up post hoc analyses (say by using Dunn's test (Dinno 2015) or Tukey's honestly significant difference (HSD) test (Vogt 2015)) typically rank algorithms, shedding light on which pairs show statistically significant differences. It must be stressed that clear guidelines regarding the number of independent runs, significance levels, and effect sizes vary in consistency in the broader research community. While a thorough statistical validation process may confirm or refute the efficacy of a new method, it can also direct attention to specific problem instances where performance anomalies arise.

5.3 | Visualization Techniques

While numeric metrics and statistical tests quantify performance, visualization offers intuitive insights into an algorithm's behavior (Lotif 2014)—see Figure 4. In continuous optimization with low-dimensional benchmarks, contour plots or surface topographies can depict the landscape's structure and highlight basins of attraction, saddle points, or ridges. Plotting an algorithm's search trajectory onto this contour reveals how different steps progress or stall in specific regions (Osaba et al. 2021). If the search path skirts around a steep slope or meanders within a deceptive valley, the visualization can pinpoint weaknesses in the update rules or the balance between exploration and exploitation (Halim et al. 2006).

Fitness-distance correlation plots examine relationships between candidate solutions' fitness values and their distances to known optima (Li, Sun, et al. 2022). Such visualization can indicate whether populations cluster around local minima or if they steadily refine toward the best region. These visualizations can also help infer the degree of diversity maintained throughout the run. For example, a population that collapses too quickly toward a suboptimal cluster may struggle to relocate the global optimum (Wang et al. 2018). In contrast, an overly dispersed set might waste computational effort without honing in on promising basins.

Furthermore, trajectory plots can further disclose fluctuations in objective values over iterations to diagnose whether an algorithm steadily improves, oscillates around particular values, or degrades under certain conditions (Ochoa et al. 2021). Despite reduced dimensionality and simplified displays, these charts help identify phases where optimization progress stalls/improves. Such observations can help promote a deeper investigation into potential parameter adjustments or synergy with other components like learning rates or mutation rates.

In multi-objective optimization, scatter plots depicting the distribution of solutions in objective space can illustrate how completely an algorithm explores the Pareto front (Talbi et al. 2012). For example, dense clustering in certain regions might indicate local dominance of that objective trade-off, while sparse coverage elsewhere reveals potential blind spots. Tracking solutions spread over multiple iterations can, therefore, clarify whether the method incrementally discovers

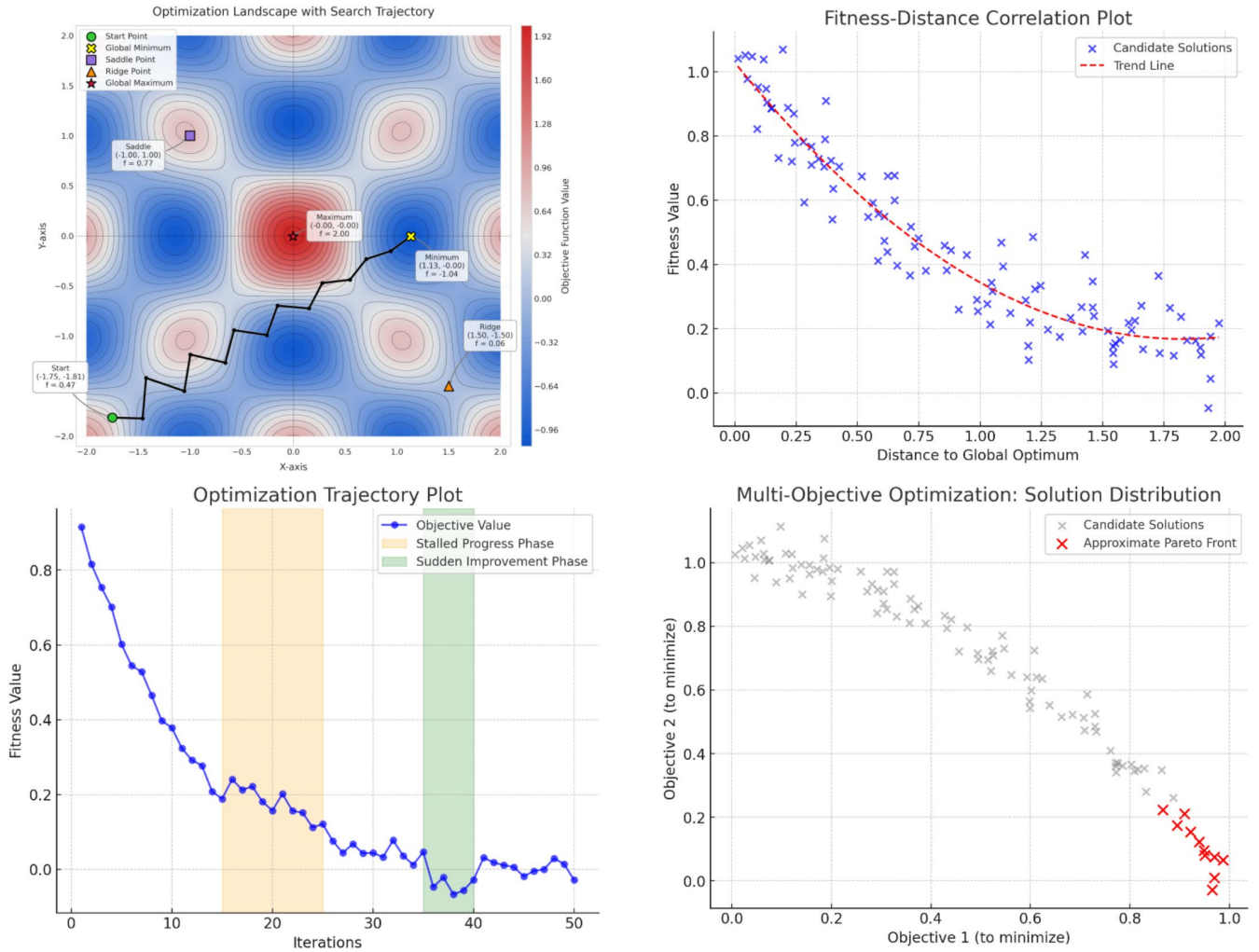

FIGURE 4 | Illustration of typical visualization methods.

better trade-offs or remains stuck in partial regions of the frontier (Agrawal et al. 2008). For dynamic or time-varying benchmarks, animated visualizations can demonstrate how quickly the algorithm adapts to shifts in the landscape, underscoring whether populations reposition themselves effectively or lag behind moving optima.

6 | Best Practices and Framework for Benchmarking

As seen from the above review, selecting suitable benchmark functions demands thoughtful alignment between algorithmic objectives, domain-specific requirements, and the distinctive features of potential test problems. The following discussion provides a technically oriented perspective on designing and carrying out meaningful benchmarking and a conceptual framework that users can adopt/adapt.

6.1 | A Discussion on Best Practices

The first consideration in any benchmarking campaign is identifying the algorithmic properties a developer is looking to investigate (Naser 2023; Vermetten et al. 2024). For example, a developer may propose a method that exploits smoothness in the objective function or gracefully handles large-scale uncertainty. Suppose a new algorithm relies heavily on local gradient approximations or uses a population-based mechanism to escape shallow basins. In that case, the performance metrics of interest will differ from those for a purely gradient-based method that assumes precise derivative information. Moreover, domain-specific constraints, or the possibility of dynamic changes in the objective landscape, might determine whether an algorithm can adapt quickly or maintain stable performance over time (Du and Li 2008). Subsequently, it is prudent to articulate clearly, at the very start, the types of problem structures or search dynamics that the benchmark set should highlight (Dangerfield and Roberts 1996). This initial clarity on algorithmic properties prevents the inadvertent selection of test problems that either fail to test critical functionalities or produce distorted impressions of performance.

Once the primary traits of the algorithm have been identified, the next major task is to assemble a suite of problems that examine these traits so that each element of the test suite should serve a distinct purpose (Simon 2013). For example, if an algorithm is purported to handle extreme multimodality

without additional constraints, a developer (or benchmarker) might want to include problems with several local traps or basin shapes. If the algorithm's strength lies in coping with constraints, that suite might include problems featuring intricate feasible regions or discontinuities that break naive search strategies. The overarching objective is to create a set of test conditions in which each problem reveals something different about how the algorithm operates (Beiranvand et al. 2017; Polak 2012). For instance, a method that possesses efficient searching in low-dimensional continuous spaces might falter when confronted with higher-dimensional versions of the same class of problems. Such expansion to higher dimensions can reveal scaling issues hidden in simpler tasks. On the contrary, if the algorithm claims to be robust to perturbations/noise, artificially injecting varying noise levels into some of the selected functions can test whether performance degrades or remains stable.

It is equally important to consider how domain constraints can be realistically modeled. In many engineering/industrial contexts, variables are bound by practical limits that smooth bounding conditions cannot fully capture. For example, in mechanical designs, the geometry might impose non-linear constraints that interact subtly to result in highly specialized feasible regions. In this scenario, a purely theoretical function with a simple box constraint might not fully approximate these realities to draw meaningful conclusions (Hager and Zhang 2006; Deb and Jain 2014). To address this item, at least a portion of the benchmarking suite should reflect scenarios derived from actual application domains (Amaran et al. 2016). This might mean scaling down the dimensionality or restricting the range of the design variables so that the problem remains tractable (but the essential structure of the constraints is still retained).

A recurring theme in benchmarking is the tension between coverage and redundancy (Aoues and Chateauneuf 2010). On the one hand, it is desirable to have a broad range of benchmarks that capture specific features of landscapes (e.g., smoothness and constraints). On the other hand, a suite that is too large or lacks a clear rationale for each component might dilute the meaningfulness of results (Bynum et al. 2021). It is, therefore, valuable to trim the test set to a representative subset where each problem sheds light on a unique performance aspect. In addition to simplifying the interpretation of findings, a carefully selected but relatively concise suite makes it feasible to conduct thorough multiple trials and deeper analysis of each result (Hansen et al. 2014).

Another dimension revolves around the methodology for measuring and reporting performance metrics (Mattos et al. 2021). Results can become tangled without a systematic approach to measurements, especially if only the final solution quality is reported without regard to the time, settings, or number of evaluations required. Similarly, focusing solely on the best results from a small number of runs can give an inflated sense of an algorithm's capabilities; a more rigorous approach involves reporting statistical measures that capture both central tendencies and outliers. Beyond the raw performance metrics, the experimental design can also significantly influence benchmarking reliability (Pham and Castellani 2014). For example, determining how many independent runs are conducted, how random seeds are managed, whether the algorithm is re-initialized completely from scratch each time, and how results are aggregated all impact the final results. In many cases, the observed differences in performance between algorithms might be small, so statistical rigor becomes essential (Hansen et al. 2010). Applying confidence intervals, hypothesis testing, or non-parametric statistical methods can help confirm observed differences.

A further complexity arises from tuning algorithmic parameters (Grefenstette 1986). If these parameters are not chosen systematically, results may be heavily skewed in favor of whichever configuration works best on a given problem instance (Yang et al. 2013). Thus, a more precise strategy includes specifying the parameter choices a priori based on well-reasoned arguments or implementing automated parameter tuning approaches to find robust settings across the entire suite. Reporting the method by which parameters were selected, including any baseline comparisons against default configurations, fosters transparency (Eiben and Smit 2011).

When presenting final results, an essential best practice is distinguishing between convergence and final outcome (Zhang et al. 2004). Reporting one without the other offers only a partial picture since an algorithm may converge rapidly but fail to reach a high-quality solution, or it may eventually achieve an excellent solution only after many evaluations. For time-sensitive domains, the former might be more relevant; for precision-critical applications, the latter might dominate.

Another aspect of benchmarking that deserves special mention is the role of parallelization, high-performance computing (HPC), and memory usage (Wang et al. 2022). Many modern optimization algorithms can exploit parallel resources to evaluate multiple candidate solutions simultaneously or to run separate threads to explore different regions of the search space. If the benchmarking environment includes such parallelization, it is important to be transparent about how computational resources are allocated and measured. Reporting parallel efficiency (the ratio between speed-up and the number of computational units) can help isolate whether performance gains are due to algorithmic design or concurrency. In some cases, an algorithm might scale poorly as the number of processors increases, negating any theoretical advantage in large-scale HPC environments (Ezekiel et al. 2011). Similarly, suppose an algorithm maintains a large population of candidate solutions or relies on memory-intensive structures to model gradients or constraints. In that case, it might consume more resources than an alternative approach with a leaner memory footprint.

While the discussion so far has largely addressed single-objective scenarios, it is important to note that applications require balancing multiple goals that may conflict with each other. Thus, the selection of test problems in multi- or many-objective settings should account for problem features such as convex versus non-convex Pareto fronts, discontinuities, degeneracies, or a high number of objectives (Tusar and Filipic 2015). Ensuring that the chosen problems capture these attributes meaningfully ensures that one does not overclaim generality based on overly simplistic tasks. While the

core principles of selecting diverse problem features, assessing domain relevance, and adopting measurement and analysis techniques remain consistent, multi- or many-objective benchmarking adds another layer of complexity that must be approached with care.

6.2 | Framework

The above considerations can be combined to conceptualize a possible framework for benchmarking. Initially, the developer defines the scope of the optimization algorithm: whether it is single-objective or multi- or many-objective, unconstrained or constrained, continuous or discrete, static or dynamic, low-dimensional or high-dimensional, or a combination of these features. This scope determination sets the boundaries for selecting appropriate test problems. The next stage involves mapping each interest feature to at least one or two problems in the test suite. This mapping ensures a direct correspondence between the algorithm's stated capabilities (or intended use cases) and the challenges embedded in the benchmarks. Once this mapping is complete, the benchmarker can finalize a minimal but comprehensive set of problems that collectively span the performance space (see Figure 5).

The third stage in the framework is establishing a performance evaluation protocol, which includes selecting metrics (such as best-so-far objective value, average objective value, standard deviation, or advanced multi-objective indicators) and specifying the maximum number of function evaluations or the computational budget. This stage also formalizes whether the comparison will rely on parallelization, how runs will be repeated and aggregated statistically, and how parameter tuning will be managed to avoid artificially inflating performance. The importance of random seeds and thorough documentation of algorithmic settings for both validation and consistency is emphasized. The framework's fourth stage is actual execution, where the algorithm(s) are run on each selected benchmark problem under predefined conditions. All relevant data are recorded for subsequent analysis to maintain consistency and guard against unconscious bias.

The final stage in this proposed framework is a rigorous post-processing and reporting analysis. This includes computing statistical measures, generating convergence profiles, comparing outcomes, and possibly dissecting the algorithm's behavior at different stages of its search process (for instance, analyzing how quickly it escapes local minima or how well it adapts when the environment changes). At this point, the developer should return to the initial design goals, verifying whether the empirical evidence supports the algorithm's advertised capabilities. If unexpected weaknesses or inconsistencies are revealed, these results should be presented candidly, as they provide valuable guidance for future refinements. Partial success or failure in specific benchmark categories can often be more instructive than a purely positive outcome since it pinpoints where the algorithm might need further development.

In addition to the proposed framework, discussing how benchmarkers decide to handle partial success in a benchmark can be of merit. For example, an algorithm might not reach the global optimum but could converge to a useful solution more consistently than another approach (Li, Zhan, et al. 2022). Depending on the domain, partial improvements might be acceptable or preferable if they come with significantly lower computational costs or simpler parameter requirements (Rahi et al. 2021). Reporting these notes ensures practitioners who read the benchmarking results can decide whether a method meets their needs. This is especially relevant for resource-constrained scenarios where near-optimal solutions delivered quickly might be more valuable than optimal solutions that take substantially longer or require specialized hardware.

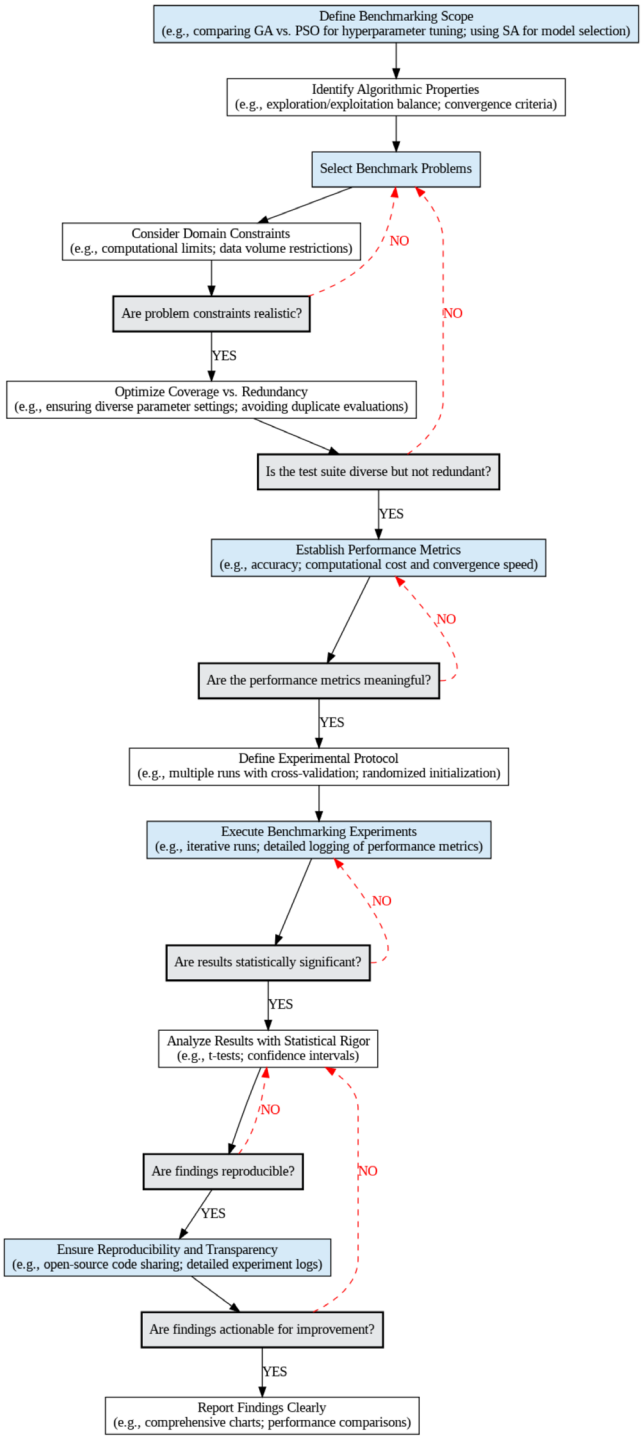

FIGURE 5 | Proposed flowchart for benchmarking.

As optimization algorithms evolve, a good benchmark suite can be periodically refined to reflect new insights, changing hardware capabilities, and evolving problem domains (Vigo-Aguiar and Alonso 2013). Longitudinal benchmarking, in which a method is tested on the same suite multiple times as the method or computing environment changes, can reveal how design improvements affect performance and whether certain modifications genuinely advance state of the art or merely shift performance from one metric to another (Välikangas et al. 2022). This continuous evolution of benchmarking practices underlines the necessity of flexible frameworks, new metrics, and new computational paradigms.

7 | Challenges, Limitations, and Future Directions

Despite ongoing efforts to craft design benchmarks, significant challenges persist in optimization research (Alba et al. 2013; Floudas et al. 2004). This section examines some of the common pitfalls (challenges) developers encounter when using benchmark functions and sheds light on future research directions.

For a start, selective reporting can lead to performance claims that fail to generalize beyond the chosen test functions (Sarhani et al. 2023). Thus, overfitting becomes a practical risk because it does not address diverse obstacles. Another challenge concerns the lack of standardization (Velasco et al. 2024). Parameter-setting discrepancies can produce contradictory outcomes even when using the same test functions (Shadkam 2022). This inconsistency complicates direct comparisons and undermines reproducibility. Further, the absence of standardized rules for the number of runs, initial conditions, function evaluation budgets, or statistical tests intensifies this challenge. Beyond reproducibility, overemphasis on classical functions can lead to stagnation in technique refinement since such functions often rely on particular landscapes (and may not capture discrete natures) (Razumikhin 1987).

One substantial shortcoming is the underrepresentation of complexity (Bubeck 2015). Although some specialized test suites incorporate dynamic elements/constraints/noisy evaluations, these features often remain somewhat simplified and do not replicate time-series variations or anomalies. Similarly, dynamic benchmarks frequently rely on smoothly evolving landscapes, even though real-world scenarios may undergo abrupt transitions (Yazdani et al. 2022). While some established dynamic suites exist, future work will likely intensify their complexity to shed light on how algorithms respond to major disruptions (Yazdani et al. 2021a, 2021b).

Large-scale and high-dimensional benchmarks remain unevenly addressed (Esmin et al. 2015). Although researchers can scale existing functions to higher dimensions, this approach does not always reflect realistic parameter correlations or data dependencies. Expanding benchmark suites to include truly high-dimensional domains with correlated variables, multiple scales, or region-specific constraints poses another unique challenge (Yang et al. 2018). Another challenge tied to this issue requires refining how performance is measured in high dimensions and under varying scales. Traditional metrics such as final solution accuracy or convergence speed might mask underlying challenges in exploration. Developers could integrate problem-specific measures that evaluate how well the search process identifies relevant subspaces around critical variables (Wang et al. 2019).

While much of the current focus remains on refining classical test suites and bridging the gap between synthetic and real-world tasks, future directions are invited to expand the breadth and depth of benchmark-driven research. For example, quantum computing has begun to influence the field of optimization, primarily through quantum-inspired methods that borrow principles from quantum mechanics without requiring fully quantum hardware (Li et al. 2020). Such algorithms can potentially solve specific problems with increased efficiency (Moll et al. 2018). For example, combining evolutionary operators with quantum principles can yield new strategies for escaping local optima. Capturing these synergies in test functions requires unique design choices (Cano et al. 2017).

At present, test functions are often treated as black boxes, with limited emphasis on clarifying which factors most influence the solution space (Audet 2014; Alarie et al. 2021). With the rise of explainability, future benchmarks could incorporate interpretable structures to enable researchers to identify which variables or constraints drive the shape of the fitness landscape (Naser 2021). These functions could help resolve the rationale behind an algorithm's decisions by embedding meta-information or hierarchical relationships. This added layer of transparency fosters a better understanding of why certain methods excel (Zhou et al. 2024). For instance, an explainable function might label key features or document how a given parameter changes in a specific domain region affect the global optimum (Fyvie et al. 2025). This supplementary context can help diagnose algorithmic failures and improve generalizability.

Strategies to mitigate many of the aforementioned challenges and limitations revolve around community initiatives to encourage the open sharing of new test functions, data sets, and code repositories (Nejati et al. 2023; Khari et al. 2020). Such an approach would foster incremental improvements in the representativeness of benchmark suites. Therefore, the co-evolution of benchmark suites that combine discrete subproblems with continuous subproblems might highlight the adaptability of hybrid optimization algorithms (Kumar et al. 2021; Van Thieu and Mirjalili 2023). Further, emerging technologies for containerization (e.g., Docker (Combe et al. 2016)) can help streamline the process of reproducing experimental environments (Casalicchio and Iannucci 2020). Such community-driven efforts might also involve crowd-sourced feedback on potential new benchmarks or modifications to existing ones.

8 | Conclusions

This review has surveyed the landscape of benchmark and test functions with respect to evaluating ML optimization methods and metaheuristics. The present discussion covers classical benchmarks that have long served as foundational stress tests and several methodologies that guide fair comparisons. Special attention is paid to key function attributes, such as modality, separability, and the presence of noise, for a number of the

commonly used benchmark functions. In addition, this review highlights the best practices for evaluating performance and proposes a framework for selecting test functions. The interplay of emerging trends (such as quantum-inspired optimization, explainable test functions, and community-driven benchmarking platforms) suggests a productive ground for innovation as a means to integrate real-world complexity and embrace reproducible research norms. In addition:

- Relying on a limited set of well-known benchmarks can lead to biased evaluations and inflated performance claims.
- Evaluation should encompass various metrics beyond just solution quality, including convergence speed, computational cost, and scalability.
- Best practices in benchmarking involve carefully selecting diverse problem suites, realistically modeling domain constraints, and transparently reporting results.
- Future research should focus on developing more complex, dynamic, high-dimensional, and explainable benchmark functions, along with fostering community-driven standardization.

Author Contributions

M. Z. Naser: conceptualization (equal), supervision (equal), writing – review and editing (equal). **Mohammad Khaled Al-Bashiti:** investigation (equal), writing – original draft (equal). **Arash Teymori Gharah Tapeh:** data curation (equal), writing – original draft (equal). **Ahmad Naser:** conceptualization (equal), writing – original draft (equal). **Venkatesh Kodur:** writing – original draft (equal). **Rami Hawileh:** writing – original draft (equal). **Jamal Abdalla:** writing – original draft (equal). **Nima Khodadadi:** writing – original draft (equal). **Amir H. Gandomi:** writing – original draft (equal). **Armin Dadras Eslamlou:** methodology (equal), writing – original draft (equal).

Conflicts of Interest

The authors declare no conflicts of interest.

Data Availability Statement

Data is available on request from the author. The list of optimization functions collected through this review can be found at: https://github.com/mznaser-clemson/Optimization_functions/tree/main.

Related WIREs Articles

Benchmarking in classification and regression

Critical review of bio-inspired optimization techniques

References

Abdel-Basset, M., R. Mohamed, M. B. Jasser, I. M. Hezam, k. M. Sallam, and A. W. Mohamed. 2023. "Developments on Metaheuristic-Based Optimization for Numerical and Engineering Optimization Problems: Analysis, Design, Validation, and Applications." *Alexandria Engineering Journal* 78: 175–212. https://doi.org/10.1016/j.aej.2023.07.039.

Adby, P. 2013. *Introduction to Optimization Methods*. Springer.

Agrawal, S., Y. Dashora, M. Tiwari, and Y. J. Son. 2008. "Interactive Particle Swarm: A Pareto-Adaptive Metaheuristic to Multiobjective Optimization." *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans* 38, no. 2: 258–277. https://doi.org/10.1109/TSMCA.2007.914767.

Agrawal, T. 2020. *Hyperparameter Optimization in Machine Learning: Make Your Machine Learning and Deep Learning Models More Efficient*, 1–166. Apress. https://doi.org/10.1007/978-1-4842-6579-6.

Agushaka, J. O., O. Akinola, A. E. Ezugwu, O. N. Oyelade, and A. K. Saha. 2022. "Advanced Dwarf Mongoose Optimization for Solving CEC 2011 and CEC 2017 Benchmark Problems." *PLoS One* 17: e0275346. https://doi.org/10.1371/journal.pone.0275346.

Alarie, S., C. Audet, A. E. Gheribi, M. Kokkolaras, and S. le Digabel. 2021. "Two Decades of Blackbox Optimization Applications." *EURO Journal on Computational Optimization* 9: 100011. https://doi.org/10.1016/j.ejco.2021.100011.

Alba, E., G. Luque, and S. Nesmachnow. 2013. "Parallel Metaheuristics: Recent Advances and New Trends." *International Transactions in Operational Research* 20: 1–48. https://doi.org/10.1111/j.1475-3995.2012.00862.x.

Ali, M. M., C. Khompatraporn, and Z. B. Zabinsky. 2005. "A Numerical Evaluation of Several Stochastic Algorithms on Selected Continuous Global Optimization Test Problems." *Journal of Global Optimization* 31: 635–672. https://doi.org/10.1007/s10898-004-9972-2.

Alimoradi, A., C. M. Foley, and S. Pezeshk. 2010. "Benchmark Problems in Structural Design and Performance Optimization: Past, Present, and Future Part I." Proceedings of the 19th Analysis and Computation Specialty Conference. https://doi.org/10.1061/41131(370)40.

Amaran, S., N. V. Sahinidis, B. Sharda, and S. J. Bury. 2016. "Simulation Optimization: A Review of Algorithms and Applications." *Annals of Operations Research* 240: 351–380. https://doi.org/10.1007/s10479-015-2019-x.

Andrei, N. 2008. "An Unconstrained Optimization Test Functions Collection." In *Advanced Modelling and Optimization*. Center for Advanced Modeling and Optimization.

Anil, R., V. Gupta, T. Koren, and Y. Singer. 2019. "Memory-Efficient Adaptive Optimization." In *Advances in Neural Information Processing Systems*. NeurIPS Proceedings.

Aoues, Y., and A. Chateauneuf. 2010. "Benchmark Study of Numerical Methods for Reliability-Based Design Optimization." *Structural and Multidisciplinary Optimization* 41: 277–294. https://doi.org/10.1007/s00158-009-0412-2.

Audet, C. 2014. "A Survey on Direct Search Methods for Blackbox Optimization and Their Applications." In *Mathematics Without Boundaries Surveys in Interdisciplinary Research*. Springer. https://doi.org/10.1007/978-1-4939-1124-0_2.

Bartz-Beielstein, T., M. Chiarandini, L. Paquete, and M. Preuss. 2010. *Experimental Methods for the Analysis of Optimization Algorithms*. Springer. https://doi.org/10.1007/978-3-642-02538-9.

Beiranvand, V., W. Hare, and Y. Lucet. 2017. "Best Practices for Comparing Optimization Algorithms." *Optimization and Engineering* 18: 815–848. https://doi.org/10.1007/s11081-017-9366-1.

Ben Younes, E., C. Changenet, J. Bruyère, E. Rigaud, and J. Perret-Liaudet. 2022. "Multi-Objective Optimization of Gear Unit Design to Improve Efficiency and Transmission Error." *Mechanism and Machine Theory* 167: 104499. https://doi.org/10.1016/j.mechmachtheory.2021.104499.

Beyer, H. G., and B. Sendhoff. 2007. "Robust Optimization - A Comprehensive Survey." *Computer Methods in Applied Mechanics and Engineering* 196: 3190–3218. https://doi.org/10.1016/j.cma.2007.03.003.

Bonyadi, M. R., and Z. Michalewicz. 2017. "Particle Swarm Optimization for Single Objective Continuous Space Problems: A Review." *Evolutionary Computation* 25: 1–54. https://doi.org/10.1162/EVCO_r_00180.

Boussaïd, I., J. Lepagnot, and P. Siarry. 2013. "A Survey on Optimization Metaheuristics." *Information Sciences* 237: 82–117. https://doi.org/10.1016/j.ins.2013.02.041.

Brockhoff, D., T. do Tran, and N. Hansen. 2015. "Benchmarking Numerical Multiobjective Optimizers Revisited." GECCO'15: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. https://doi.org/10.1145/2739480.2754777.

Brownlee, J. 2007. "A Note on Research Methodology and Benchmarking Optimization Algorithms." Complex Intelligent Systems Laboratory (CIS), Centre for Information Technology Research (CITR), Faculty of Information and Communication Technologies (ICT), Swinburne University of Technology, Victoria, Australia, Technical Report ID.

Bubeck, S. 2015. "Convex Optimization: Algorithms and Complexity." *Foundations and Trends in Machine Learning* 8: 231–357. https://doi.org/10.1561/2200000050.

Bull, A. D. 2011. "Convergence Rates of Efficient Global Optimization Algorithms." *Journal of Machine Learning Research* 12.

Bynum, M. L., G. A. Hackebeil, W. E. Hart, et al. 2021. *Pyomo — Optimization Modeling in Python*. Vol. 67. Springer. https://doi.org/10.1007/978-3-030-68928-5.

Cano, A., C. García-Martínez, and S. Ventura. 2017. "Extremely High-Dimensional Optimization With MapReduce: Scaling Functions and Algorithm." *Information Sciences* 415-416: 110–127. https://doi.org/10.1016/j.ins.2017.06.024.

Cárdenas-Montes, M., M. A. Vega-Rodríguez, J. J. Rodríguez-Vázquez, and A. Gómez-Iglesias. 2015. "A Comparison Exercise on Parallel Evaluation of Rosenbrock Function." Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation. https://doi.org/10.1145/2739482.2764641.

Casalicchio, E., and S. Iannucci. 2020. "The State-Of-The-Art in Container Technologies: Application, Orchestration and Security." *Concurrency and Computation: Practice and Experience* 32. https://doi.org/10.1002/cpe.5668.

Cavazzuti, M. 2013. *Optimization Methods: From Theory to Design Scientific and Technological Aspects in Mechanics*, 1–262. Springer. https://doi.org/10.1007/978-3-642-31187-1/COVER.

Chandra, R., R. Deo, K. Bali, and A. Sharma. 2016. "On the Relationship of Degree of Separability With Depth of Evolution in Decomposition for Cooperative Coevolution." Proceedings of the 2016 IEEE Congress on Evolutionary Computation (CEC). https://doi.org/10.1109/CEC.2016.7744408.

Chen, T., X. Chen, W. Chen, et al. 2022. "Learning to Optimize: A Primer and A Benchmark." *Journal of Machine Learning Research* 23: 1–59.

Cheng, R., Y. Jin, M. Olhofer, and B. Sendhoff. 2017. "Test Problems for Large-Scale Multiobjective and Many-Objective Optimization." *IEEE Transactions on Cybernetics* 47: 4108–4121. https://doi.org/10.1109/TCYB.2016.2600577.

Chinneck, J. W. 2008. *Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods*, 270. Springer.

Chopard, B., and M. Tomassini. 2018. "An Introduction to Metaheuristics for Optimization." *Nature Computational Science*, 226. Springer.

Combe, T., A. Martin, and R. di Pietro. 2016. "To Docker or Not to Docker: A Security Perspective." *IEEE Cloud Computing* 3: 54–62. https://doi.org/10.1109/MCC.2016.100.

Cuevas, E., A. Luque, B. Morales Castañeda, and B. Rivera. 2024. *Introduction to Metaheuristic Methods*, 1–10. Springer. https://doi.org/10.1007/978-3-031-63053-8_1.

Cui, S., A. M. C. So, and R. Zhang. 2017. "Unconstrained and Constrained Optimization Problems." In *Mathematical Foundations for Signal Processing, Communications, and Networking*. CRC Press. https://doi.org/10.1201/9781351105668.

Dangerfield, B., and C. Roberts. 1996. "An Overview of Strategy and Tactics in System Dynamics Optimization." *Journal of the Operational Research Society* 47: 405–423. https://doi.org/10.1057/jors.1996.40.

Daoud, M. S., M. Shehab, H. M. Al-Mimi, L. Abualigah, R. A. Zitar, and M. K. Y. Shambour. 2023. "Gradient-Based Optimizer (GBO): A Review, Theory, Variants, and Applications." *Archives of Computational Methods in Engineering* 30: 2431–2449. https://doi.org/10.1007/s11831-022-09872-y.

David, M., K. Sigrid, J. M. Frayret, P. Gilles, and G. Nicolas. 2015. "A Review and Taxonomy of Interactive Optimization Methods in Operations Research." *ACM Transactions on Interactive Intelligent Systems* 5: 1–43. https://doi.org/10.1145/2808234.

Deb, K., and H. Gupta. 2006. "Introducing Robustness in Multi-Objective Optimization." *Evolutionary Computation* 14: 463–494. https://doi.org/10.1162/evco.2006.14.4.463.

Deb, K., and H. Jain. 2014. "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints." *IEEE Transactions on Evolutionary Computation* 18: 577–601. https://doi.org/10.1109/TEVC.2013.2281535.

Deb, K., A. Sinha, and S. Kukkonen. 2006. "Multi-Objective Test Problems, Linkages, and Evolutionary Methodologies." Proceedings of the GECCO 2006 Genetic and Evolutionary Computation Conference. https://doi.org/10.1145/1143997.1144179.

Dieterich, J. M., and B. Hartke. 2012. "Empirical Review of Standard Benchmark Functions Using Evolutionary Global Optimization." *Applied Mathematics* 03: 1552–1564. https://doi.org/10.4236/am.2012.330215.

Dinno, A. 2015. "Nonparametric Pairwise Multiple Comparisons in Independent Groups Using Dunn's Test." *Stata Journal* 15: 292–300. https://doi.org/10.1177/1536867x1501500117.

Diwekar, U. M. 2021. *Introduction to Applied Optimization*. Springer.

Dobslaw, F. 2010. *A Parameter-Tuning Framework for Metaheuristics Based on Design of Experiments and Artificial Neural Networks*. World Academy of Science, Engineering and Technology.

Droste, S., T. Jansen, and I. Wegener. 1998. "On the Optimization of Unimodal Functions With the (1 + 1) Evolutionary Algorithm." In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer. https://doi.org/10.1007/bfb0056845.

Du, W., and B. Li. 2008. "Multi-Strategy Ensemble Particle Swarm Optimization for Dynamic Optimization." *Information Sciences* 178: 3096–3109. https://doi.org/10.1016/j.ins.2008.01.020.

Durillo, J. J., A. J. Nebro, C. A. C. Coello, J. Garcia-Nieto, F. Luna, and E. Alba. 2010. "A Study of Multiobjective Metaheuristics When Solving Parameter Scalable Problems." *IEEE Transactions on Evolutionary Computation* 14: 618–635. https://doi.org/10.1109/TEVC.2009.2034647.

Durillo, J. J., A. J. Nebro, F. Luna, C. A. Coello Coello, and E. Alba. 2010. "Convergence Speed in Multi-Objective Metaheuristics: Efficiency Criteria and Empirical Study." *International Journal for Numerical Methods in Engineering* 84: 1344–1375. https://doi.org/10.1002/nme.2944.

ECAI. 2006. *Subtitle of Host Publication 17th European Conference on Artificial Intelligence August 29 - September 1, 2006, Riva del Garda, Italy*, edited by Gerhard Brewka, Silvia Coradeschi, Anna Perini, and Paolo Traverso. Publication series. Frontiers in Artificial Intelligence and Applications. vol. 141, 703–704. IOS Press BV.

Eiben, A. E., and S. K. Smit. 2011. "Parameter Tuning for Configuring and Analyzing Evolutionary Algorithms." *Swarm and Evolutionary Computation* 1: 19–31. https://doi.org/10.1016/j.swevo.2011.02.001.

Emiola, I., and R. Adem. 2021. "Comparison of Minimization Methods for Rosenbrock Functions." Proceedings of the 2021 29th Mediterranean Conference on Control and Automation (MED). https://doi.org/10.1109/MED51440.2021.9480200.

Esmin, A. A. A., R. A. Coelho, and S. Matwin. 2015. "A Review on Particle Swarm Optimization Algorithm and Its Variants to Clustering High-Dimensional Data." *Artificial Intelligence Review* 44: 23–45. https://doi.org/10.1007/s10462-013-9400-4.

Ezekiel, J., G. Lüttgen, and R. Siminiceanu. 2011. "To Parallelize or to Optimize." *Journal of Logic and Computation* 21: 85–120. https://doi.org/10.1093/logcom/exp006.

Farag, M. A., M. A. El-Shorbagy, A. A. Mousa, and I. M. El-Desoky. 2020. "A New Hybrid Metaheuristic Algorithm for Multiobjective Optimization Problems." *International Journal of Computational Intelligence Systems* 13: 920–940. https://doi.org/10.2991/ijcis.d.200618.001.

Floudas, C. A., I. G. Akrotiriankis, S. Caratzoulas, C. A. Meyer, and J. Kallrath. 2004. "Global Optimization in the 21st Century: Advances and Challenges." *Computer Aided Chemical Engineering* 18: 23–51. https://doi.org/10.1016/S1570-7946(04)80082-8.

Fonseca, R. M., E. Della Rossa, A. A. Emerick, R. G. Hanea, and J. D. Jansen. 2020. "Introduction to the Special Issue: Overview of OLYMPUS Optimization Benchmark Challenge." *Computational Geosciences* 24: 1933–1941. https://doi.org/10.1007/s10596-020-10003-4.

Fox, M., S. Yang, and F. Caraffini. 2022. "A New Moving Peaks Benchmark With Attractors for Dynamic Evolutionary Algorithms." *Swarm and Evolutionary Computation* 74: 101125. https://doi.org/10.1016/j.swevo.2022.101125.

Fyvie, M., J. A.W. Mccall, L. A. Christie, et al. 2025. "Towards Explainable Metaheuristics: Feature Extraction From Trajectory Mining." *Expert Systems* 42: e13494. https://doi.org/10.1111/EXSY.13494.

Gandomi, A. H., and X. S. Yang. 2011. *Benchmark Problems in Structural Optimization, Studies in Computational Intelligence*. Springer. https://doi.org/10.1007/978-3-642-20859-1_12.

Grefenstette, J. J. 1986. "Optimization of Control Parameters for Genetic Algorithms." *IEEE Transactions on Systems, Man, and Cybernetics* 16, no. 1: 122–128. https://doi.org/10.1109/TSMC.1986.289288.

Gunantara, N. 2018. "A Review of Multi-Objective Optimization: Methods and Its Applications." *Cogent Engineering* 5: 1502242. https://doi.org/10.1080/23311916.2018.1502242.

Gupta, S., H. Abderazek, B. S. Yıldız, A. R. Yildiz, S. Mirjalili, and S. M. Sait. 2021. "Comparison of Metaheuristic Optimization Algorithms for Solving Constrained Mechanical Design Optimization Problems." *Expert Systems with Applications* 183: 115351. https://doi.org/10.1016/j.eswa.2021.115351.

Hager, W. W., and H. Zhang. 2006. "A New Active Set Algorithm for Box Constrained Optimization." *SIAM Journal on Optimization* 17: 526–557. https://doi.org/10.1137/050635225.

Halim, S., R. H. C. Yap, and H. C. Lau. 2006. "Visualization for Analyzing Trajectory-Based Metaheuristic Search Algorithms." In *Frontiers in Artificial Intelligence and Applications*, edited by G. Brewka, S. Coradeschi, A. Perini, and P. Traverso, vol 141,703–704. IOS Press.

Hansen, N., A. Auger, S. Finck, and R. Ros. 2010. "Real-Parameter Black-Box Optimization Benchmarking 2009: Experimental Setup."

Hansen, N., A. Auger, S. Finck, and R. Ros. 2014. "Real-Parameter Black-Box Optimization Benchmarking: Experimental Setup, Report."

Hansen, P., B. Jaumard, and S. H. Lu. 1992. "Global Optimization of Univariate Lipschitz Functions: II. New Algorithms and Computational Comparison." *Mathematical Programming* 55: 273–292. https://doi.org/10.1007/BF01581203.

Hellwig, M., and H. G. Beyer. 2019. "Benchmarking Evolutionary Algorithms for Single Objective Real-Valued Constrained Optimization – A Critical Review." *Swarm and Evolutionary Computation* 44: 927–944. https://doi.org/10.1016/j.swevo.2018.10.002.

Hochbaum, D. S., and J. G. Shanthikumar. 1990. "Convex Separable Optimization Is Not Much Harder Than Linear Optimization." *Journal of the ACM (JACM)* 37: 843–862. https://doi.org/10.1145/96559.96597.

Hoffmann, F., T. Bertram, R. Mikut, M. Reischl, and O. Nelles. 2019. "Benchmarking in Classification and Regression." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 9. https://doi.org/10.1002/widm.1318.

Huang, C., S. Zhai, P. Guo, and J. Susskind. 2021. "MetricoPT: Learning to Optimize Black-Box Evaluation Metrics." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. https://doi.org/10.1109/CVPR46437.2021.00024.

Hussain, K., M. N. M. Salleh, S. Cheng, and R. Naseem. 2017. "Common Benchmark Functions for Metaheuristic Evaluation: A Review." *International Journal on Informatics Visualization* 1, no. 4-2: 218–223. https://doi.org/10.30630/joiv.1.4-2.65.

Intriligator, M. D. 2002. *Mathematical Optimization and Economic Theory*. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898719215.

Jamil, M., and X. S. Yang. 2013. "A Literature Survey of Benchmark Functions for Global Optimisation Problems." *International Journal of Mathematical Modelling and Numerical Optimisation* 4: 150–194. https://doi.org/10.1504/IJMMNO.2013.055204.

Jamil, M., X. S. Yang, and H. J. Zepernick. 2013. "Test Functions for Global Optimization: A Comprehensive Survey." In *Swarm Intelligence and Bio-Inspired Computation, Theory and Applications*. Elsevier. https://doi.org/10.1016/B978-0-12-405163-8.00008-9.

Kaji, T. 2021. "A Probabilistic Analysis of Neighborhoods for Combinatorial Optimization Problems and Its Application." *Journal of Heuristics* 27: 1057–1079. https://doi.org/10.1007/s10732-021-09484-y.

Kämpf, J. H., M. Wetter, and D. Robinson. 2010. "A Comparison of Global Optimization Algorithms With Standard Benchmark Functions and Real-World Applications Using EnergyPlus." *Journal of Building Performance Simulation* 3: 103–120. https://doi.org/10.1080/19401490903494597.

Kazemzadeh Azad, S., and S. Kazemzadeh Azad. 2023. "A Standard Benchmarking Suite for Structural Optimization Algorithms: ISCSO 2016–2022." *Structure* 58: 105409. https://doi.org/10.1016/j.istruc.2023.105409.

Kazikova, A., M. Pluhacek, and R. Senkerik. 2021. "How Does the Number of Objective Function Evaluations Impact Our Understanding of Metaheuristics Behavior?" *IEEE Access* 9: 44032–44048. https://doi.org/10.1109/ACCESS.2021.3066135.

Khari, M., A. Sinha, E. Verdú, and R. G. Crespo. 2020. "Performance Analysis of Six Meta-Heuristic Algorithms Over Automated Test Suite Generation for Path Coverage-Based Optimization." *Soft Computing* 24: 9143–9160. https://doi.org/10.1007/s00500-019-04444-y.

Kiseleva, E., and T. Stepanchuk. 2003. "On the Efficiency of a Global Non-Differentiable Optimization Algorithm Based on the Method of Optimal Set Partitioning." *Journal of Global Optimization*. Springer. https://doi.org/10.1023/A:1021931422223.

Kistowski, J. V., J. A. Arnold, K. Huppler, K. D. Lange, J. L. Henning, and P. Cao. 2015. "How to Build a Benchmark." ICPE 2015 – Proceedings of the 6th ACM/SPEC International Conference. https://doi.org/10.1145/2668930.2688819.

Krishnamoorthy, K. 2020. "Wilcoxon Signed-Rank Test." In *Handbook of Statistical Distributions With Applications*. Chapman and Hall/CRC. https://doi.org/10.1201/9781420011371-34.

Kumar, A., G. Wu, M. Z. Ali, et al. 2021. "A Benchmark-Suite of Real-World Constrained Multi-Objective Optimization Problems and Some Baseline Results." *Swarm and Evolutionary Computation* 67: 100961. https://doi.org/10.1016/j.swevo.2021.100961.

Lagaros, N. D., M. Kournoutos, N. A. Kallioras, and A. N. Nordas. 2023. "Constraint Handling Techniques for Metaheuristics: A State-Of-The-Art Review and New Variants." *Optimization and Engineering* 24: 2251–2298. https://doi.org/10.1007/s11081-022-09782-9.

Lan, G., J. M. Tomczak, D. M. Roijers, and A. E. Eiben. 2022. "Time Efficiency in Optimization With a Bayesian-Evolutionary Algorithm." *Swarm and Evolutionary Computation* 69: 100970. https://doi.org/10.1016/j.swevo.2021.100970.

Li, J. Y., Z. H. Zhan, and J. Zhang. 2022. "Evolutionary Computation for Expensive Optimization: A Survey." *Machine Intelligence Research* 19: 3–23. https://doi.org/10.1007/s11633-022-1317-4.

Li, W., B. Sun, Y. Huang, and S. Mahmoodi. 2022. "Adaptive Complex Network Topology With Fitness Distance Correlation Framework for Particle Swarm Optimization." *International Journal of Intelligent Systems* 37: 5217–5247. https://doi.org/10.1002/int.22790.

Li, Y., M. Tian, G. Liu, C. Peng, and L. Jiao. 2020. "Quantum Optimization and Quantum Learning: A Survey." *IEEE Access* 8: 23568–23593. https://doi.org/10.1109/ACCESS.2020.2970105.

Liang, J. J., P. N. Suganthan, and K. Deb. 2005. "Novel Composition Test Functions for Numerical Global Optimization." Proceedings 2005 IEEE Swarm Intelligence Symposium. https://doi.org/10.1109/SIS.2005.1501604.

Liang, Z., P. Zhong, M. Liu, C. Zhang, and Z. Zhang. 2022. "A Computational Efficient Optimization of Flow Shop Scheduling Problems." *Scientific Reports* 12: 845. https://doi.org/10.1038/s41598-022-04887-8.

Liu, Y., X. Ling, Z. Shi, M. Lv, J. Fang, and L. Zhang. 2011. "A Survey on Particle Swarm Optimization Algorithms for Multimodal Function Optimization." *Journal of Software* 6, no. 12: 2449–2455. https://doi.org/10.4304/jsw.6.12.2449-2455.

Loiola, E. M., N. M. M. de Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido. 2007. "A Survey for the Quadratic Assignment Problem." *European Journal of Operational Research* 176: 657–690. https://doi.org/10.1016/j.ejor.2005.09.032.

Longjohn, R., M. Kelly, S. Singh, and P. Smyth. 2024. "Benchmark Data Repositories for Better Benchmarking." *Advances in Neural Information Processing Systems* 37: 86435–86457. https://paperswithcode.com/dataset/imagenet.

López-Ibáñez, M., J. Branke, and L. Paquete. 2021. "Reproducibility in Evolutionary Computation." *ACM Transactions on Evolutionary Learning and Optimization* 1: 1–21. https://doi.org/10.1145/3466624.

Lotif, M. 2014. "Visualizing the Population of Meta-Heuristics During the Optimization Process Using Self-Organizing Maps." In *Proceedings of the 2014 IEEE Congress on Evolutionary Computation*. IEEE. https://doi.org/10.1109/CEC.2014.6900265.

Mahdavi, S., M. E. Shiri, and S. Rahnamayan. 2015. "Metaheuristics in Large-Scale Global Continues Optimization: A Survey." *Information Sciences* 295: 407–428. https://doi.org/10.1016/j.ins.2014.10.042.

Maltese, J., B. M. Ombuki-Berman, and A. P. Engelbrecht. 2018. "A Scalability Study of Many-Objective Optimization Algorithms." *IEEE Transactions on Evolutionary Computation* 22: 79–96. https://doi.org/10.1109/TEVC.2016.2639360.

Martí, R., M. Sevaux, and K. Sörensen. 2025. "Fifty Years of Metaheuristics." *European Journal of Operational Research* 321: 345–362. https://doi.org/10.1016/J.EJOR.2024.04.004.

Mattos, D. I., J. Bosch, and H. H. Olsson. 2021. "Statistical Models for the Analysis of Optimization Algorithms With Benchmark Functions." *IEEE Transactions on Evolutionary Computation* 25: 1163–1177. https://doi.org/10.1109/TEVC.2021.3081167.

Mejía-de-Dios, J.-A., and E. Mezura-Montes. 2022. "Metaheuristics: A Julia Package for Single- and Multi-Objective Optimization." *Journal of Open Source Software* 7. https://doi.org/10.21105/joss.04723.

Mirjalili, S., and A. Lewis. 2015. "Novel Performance Metrics for Robust Multi-Objective Optimization Algorithms." *Swarm and Evolutionary Computation* 21: 1–23. https://doi.org/10.1016/j.swevo.2014.10.005.

Mohamed, A. W., K. M. Sallam, P. Agrawal, A. A. Hadi, and A. K. Mohamed. 2023. "Evaluating the Performance of Meta-Heuristic Algorithms on CEC 2021 Benchmark Problems." *Neural Computing and Applications* 35: 1493–1517. https://doi.org/10.1007/s00521-022-07788-z.

Molga, M., and C. Smutnicki. 2005. "Test Functions for Optimization Needs, Test Functions for Optimization Needs."

Molina, D., A. LaTorre, and F. Herrera. 2018. "An Insight Into Bio-Inspired and Evolutionary Algorithms for Global Optimization: Review, Analysis, and Lessons Learnt Over a Decade of Competitions." *Cognitive Computation* 10: 517–544. https://doi.org/10.1007/s12559-018-9554-0.

Moll, N., P. Barkoutsos, L. S. Bishop, et al. 2018. "Quantum Optimization Using Variational Algorithms on Near-Term Quantum Devices." *Quantum Science and Technology* 3: 30503. https://doi.org/10.1088/2058-9565/aab822.

Moreau, T., M. Massias, A. Gramfort, et al. 2022. "Benchopt: Reproducible, Efficient and Collaborative Optimization Benchmarks." Advances in Neural Information Processing Systems.

Moser, I., and R. Chiong. 2013. *Dynamic Function Optimization: The Moving Peaks Benchmark. Studies in Computational Intelligence*. Springer. https://doi.org/10.1007/978-3-642-30665-5_3.

Musafer, H., E. Tokgoz, and A. Mahmood. 2022. "High-Dimensional Normalized Data Profiles for Testing Derivative-Free Optimization Algorithms." *PeerJ Computer Science* 8: e960. https://doi.org/10.7717/PEERJ-CS.960.

Naser, M. Z. 2023. *Machine Learning for Civil and Environmental Engineers: A Practical Approach to Data-Driven Analysis, Explainability, and Causality*. John Wiley & Sons.

Naser, M. Z. 2021. "An Engineer's Guide to Explainable Artificial Intelligence and Interpretable Machine Learning: Navigating Causality, Forced Goodness, and the False Perception of Inference." *Automation in Construction* 129: 103821. https://doi.org/10.1016/j.autcon.2021.103821.

Naser, M. Z., M. K. Al-Bashiti, A. Teymori, et al. 2024. "A Review of 315 Benchmark and Test Functions for Machine Learning Optimization Algorithms and Metaheuristics With Mathematical and Visual Descriptions." https://arxiv.org/abs/2406.09581v1.

Naser, M. Z., and H. Amir. 2021. "Error Metrics and Performance Fitness Indicators for Artificial Intelligence and Machine Learning in Engineering and Sciences." *Architecture, Structures and Construction* 2021, no. 1: 1–19. https://doi.org/10.1007/S44150-021-00015-8.

Nebro, A. J., J. J. Durillo, C. A. Coello Coello, F. Luna, and E. Alba. 2008. "A Study of Convergence Speed in Multi-Objective Metaheuristics." In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer. https://doi.org/10.1007/978-3-540-87700-4_76.

Nejati, F., N. A. W. Abdul Hamid, S. Z. Koohi, and Z. R. Zadeh. 2023. "An Incremental Optimization Algorithm for Efficient Verification of Graph Transformation Systems." *IEEE Access* 11: 75748–75760. https://doi.org/10.1109/ACCESS.2023.3291412.

Neumann, F., and C. Witt. 2009. "Runtime Analysis of a Simple Ant Colony Optimization Algorithm." *Algorithmica* 54: 618–627. https://doi.org/10.1007/s00453-007-9134-2.

Nissen, V., and J. Propach. 1998. "Optimization With Noisy Function Evaluations." In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer. https://doi.org/10.1007/bfb0056859.

Nocedal, J. 1992. "Theory of Algorithms for Unconstrained Optimization." *Acta Numer* 1: 199–242. https://doi.org/10.1017/S0962492900002270.

Ochoa, G., K. M. Malan, and C. Blum. 2021. "Search Trajectory Networks: A Tool for Analysing and Visualising the Behaviour of Metaheuristics." *Applied Soft Computing* 109: 107492. https://doi.org/10.1016/j.asoc.2021.107492.

Okabe, T., Y. Jin, M. Olhofer, and B. Sendhoff. 2004. "On Test Functions For Evolutionary Multi-Objective Optimization." Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). https://doi.org/10.1007/978-3-540-30217-9_80.

Olson, R. S., W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore. 2017. "PMLB: A Large Benchmark Suite for Machine Learning Evaluation and Comparison." *BioData Mining* 10: 36. https://doi.org/10.1186/s13040-017-0154-4.

Omidvar, M. N., X. Li, and X. Yao. 2021. "A Review of Population-Based Metaheuristics for Large-Scale Black-Box Global Optimization—Part I." *IEEE Transactions on Evolutionary Computation* 26, no. 5: 802–822. https://doi.org/10.1109/tevc.2021.3130838.

Omidvar, M. N., X. Li, and X. Yao. 2022. "A Review of Population-Based Metaheuristics for Large-Scale Black-Box Global Optimization—Part II." *IEEE Transactions on Evolutionary Computation* 26, no. 5: 823–843. https://doi.org/10.1109/TEVC.2021.3130835.

Osaba, E., E. Villar-Rodriguez, J. del Ser, et al. 2021. "A Tutorial on the Design, Experimentation and Application of Metaheuristic Algorithms to Real-World Optimization Problems." *Swarm and Evolutionary Computation* 64: 100888. https://doi.org/10.1016/j.swevo.2021.100888.

Öztürk, H. T., and H. T. Kahraman. 2023. "Meta-Heuristic Search Algorithms in Truss Optimization: Research on Stability and Complexity Analyses." *Applied Soft Computing* 145: 110573. https://doi.org/10.1016/j.asoc.2023.110573.

Parejo, J. A., A. Ruiz-Cortés, S. Lozano, and P. Fernandez. 2012. "Metaheuristic Optimization Frameworks: A Survey and Benchmarking." *Soft Computing* 16: 527–561. https://doi.org/10.1007/s00500-011-0754-8.

Pedregal, P. 2004. *Introduction to Optimization*, 46. Springer. https://doi.org/10.1007/B97412.

Pereira, J. L. J., G. A. Oliver, M. B. Francisco, S. S. Cunha, and G. F. Gomes. 2022. "A Review of Multi-Objective Optimization: Methods and Algorithms in Mechanical Engineering Problems." *Archives of Computational Methods in Engineering* 29, no. 4: 2285–2308. https://doi.org/10.1007/s11831-021-09663-x.

Pham, D. T., and M. Castellani. 2014. "Benchmarking and Comparison of Nature-Inspired Population-Based Continuous Optimisation Algorithms." *Soft Computing* 18: 871–903. https://doi.org/10.1007/s00500-013-1104-9.

Piotrowski, A. P., J. J. Napiorkowski, and A. E. Piotrowska. 2023. "Choice of Benchmark Optimization Problems Does Matter." *Swarm and Evolutionary Computation* 83: 101378. https://doi.org/10.1016/j.swevo.2023.101378.

Polak, E. 2012. *Optimization*. Springer.

Pop, P. C., O. Cosma, C. Sabo, and C. P. Sitar. 2024. "A Comprehensive Survey on the Generalized Traveling Salesman Problem." *European Journal of Operational Research* 314: 819–835. https://doi.org/10.1016/j.ejor.2023.07.022.

Rahi, K. H., H. K. Singh, and T. Ray. 2021. "Partial Evaluation Strategies for Expensive Evolutionary Constrained Optimization." *IEEE Transactions on Evolutionary Computation* 25: 1103–1117. https://doi.org/10.1109/TEVC.2021.3078486.

Rahimi, I., A. H. Gandomi, F. Chen, and E. Mezura-Montes. 2023. "A Review on Constraint Handling Techniques for Population-Based Algorithms: From Single-Objective to Multi-Objective Optimization." *Archives of Computational Methods in Engineering* 30: 2181–2209. https://doi.org/10.1007/s11831-022-09859-9.

Rahkar Farshi, T. 2021. "Battle Royale Optimization Algorithm." *Neural Computing and Applications* 33, no. 4: 1139–1157. https://doi.org/10.1007/s00521-020-05004-4.

Rahnamayan, S., H. R. Tizhoosh, and M. M. A. Salama. 2007. "A Novel Population Initialization Method for Accelerating Evolutionary Algorithms." *Computers and Mathematics with Applications* 53, no. 10: 1605–1614. https://doi.org/10.1016/j.camwa.2006.07.013.

Razumikhin, B. S. 1987. *Classical Principles and Optimization Problems*, 513. Springer.

Riquelme, N., C. von Lücken, and B. Barán. 2015. "Performance Metrics in Multi-Objective Optimization." Proceeding of the 2015 Latin American Computing Conference (CLEI). https://doi.org/10.1109/CLEI.2015.7360024.

Runarsson, T. P., and X. Yao. 2005. "Search Biases in Constrained Evolutionary Optimization." *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews* 35, no. 2: 233–243. https://doi.org/10.1109/TSMCC.2004.841906.

Sachs, E. 1978. "Differentiability in Optimization Theory 1." *Mathematische Operationsforschung Und Statistik. Series Optimization* 9, no. 4: 497–513. https://doi.org/10.1080/02331937808842516.

Sala, R., and R. Muller. 2020. "Benchmarking for Metaheuristic Black-Box Optimization: Perspectives and Open Challenges." 2020 IEEE Congress on Evolutionary Computation. https://doi.org/10.1109/CEC48606.2020.9185724.

Sarhani, M., S. Voß, and R. Jovanovic. 2023. "Initialization of Metaheuristics: Comprehensive Review, Critical Analysis, and Research Directions." *International Transactions in Operational Research* 30: 3361–3397. https://doi.org/10.1111/itor.13237.

Shadkam, E. 2022. "Parameter Setting of Meta-Heuristic Algorithms: A New Hybrid Method Based on DEA and RSM." *Environmental Science and Pollution Research* 29: 22404–22426. https://doi.org/10.1007/s11356-021-17364-y.

Sheldon, M. R., M. J. Fillyaw, and W. D. Thompson. 1996. "The Use and Interpretation of the Friedman Test in the Analysis of Ordinal-Scale Data in Repeated Measures Designs." *Physiotherapy Research International: The Journal for Researchers and Clinicians in Physical Therapy* 1, no. 4: 221–228. https://doi.org/10.1002/pri.66.

Shor, N. Z., A. Ruszczynski, N. Z. Kiwiel, and K. C. Shor. 2012. *Minimization Methods for Non-Differentiable Functions*. Springer.

Sigmund, O. 2022. "On Benchmarking and Good Scientific Practise in Topology Optimization." *Structural and Multidisciplinary Optimization* 65: 315. https://doi.org/10.1007/s00158-022-03427-2.

Simon, D. 2013. *Evolutionary Optimization Algorithms*. Wiley.

Skvorc, U., T. Eftimov, and P. Korosec. 2019. "CEC Real-Parameter Optimization Competitions: Progress from 2013 to 2018." 2019 IEEE Congress on Evolutionary Computation (CEC). https://doi.org/10.1109/CEC.2019.8790158.

Sra, S., S. Nowozin, and S. J. Wright. 2012. *Optimization for Machine Learning*. MIT press.

Steenkamp, C., and A. P. Engelbrecht. 2021. "A Scalability Study of the Multi-Guide Particle Swarm Optimization Algorithm to Many-Objectives." *Swarm and Evolutionary Computation* 66: 100943. https://doi.org/10.1016/j.swevo.2021.100943.

Stefanov, S. M. 2021. *Separable Optimization*, 177. Springer. https://doi.org/10.1007/978-3-030-78401-0.

Štefek, A. 2011. "Benchmarking of Heuristic Optimization Methods." Proceedings of the 14th Mechatronics, MECHATRONIKA. https://doi.org/10.1109/mechatron.2011.5961068.

Stein, O., J. Oldenburg, and W. Marquardt. 2004. "Continuous Reformulations of Discrete-Continuous Optimization Problems." *Computers and Chemical Engineering* 28, no. 10: 1951–1966. https://doi.org/10.1016/j.compchemeng.2004.03.011.

Sthle, L., and S. Wold. 1989. "Analysis of Variance (ANOVA)." *Chemometrics and Intelligent Laboratory Systems* 6, no. 4: 259–272. https://doi.org/10.1016/0169-7439(89)80095-4.

Sun, S., Z. Cao, H. Zhu, and J. Zhao. 2020. "A Survey of Optimization Methods From a Machine Learning Perspective." *IEEE Transactions on Cybernetics* 50: 3668–3681. https://doi.org/10.1109/TCYB.2019.2950779.

Suwandi, W. S. 2022. "Do Economic Growth, Income Distribution, and Investment Reduce Poverty Level?" *Jurnal Berkala Ilmiah Efisiensi* 2, no. 2: 87–96.

Talbi, E. G., M. Basseur, A. J. Nebro, and E. Alba. 2012. "Multi-Objective Optimization Using Metaheuristics: Non-Standard Algorithms." *International Transactions in Operational Research* 19: 283–305. https://doi.org/10.1111/j.1475-3995.2011.00808.x.

Tedford, N. P., and J. R. R. A. Martins. 2010. "Benchmarking Multidisciplinary Design Optimization Algorithms." *Optimization and Engineering* 11: 159–183. https://doi.org/10.1007/s11081-009-9082-6.

Terazono, Y., and A. Matani. 2015. "Continuity of Optimal Solution Functions and Their Conditions on Objective Functions." *SIAM Journal on Optimization* 25: 2050–2060. https://doi.org/10.1137/110850189.

Test Functions Index. n.d. "Test Functions Index — AMPGO 0.1.0 Documentation." https://infinity77.net/global_optimization/test_functions.html.

Tian, M., X. Yan, and X. Gao. 2024. "An Enhanced Adaptive Differential Evolution Algorithm With Dual Performance Evaluation Metrics for Numerical Optimization." *Swarm and Evolutionary Computation* 84: 101454. https://doi.org/10.1016/j.swevo.2023.101454.

Tilahun, S. L., and J. M. T. Ngnotchouye. 2017. "Firefly Algorithm for Discrete Optimization Problems: A Survey." *KSCE Journal of Civil Engineering* 21: 535–545. https://doi.org/10.1007/s12205-017-1501-1.

Tusar, T., and B. Filipic. 2015. "Visualization of Pareto Front Approximations in Evolutionary Multiobjective Optimization: A Critical Review and the Prosection Method." *IEEE Transactions on Evolutionary Computation* 19: 225–245. https://doi.org/10.1109/TEVC.2014.2313407.

Valiante, E., M. Hernandez, A. Barzegar, and H. G. Katzgraber. 2021. "Computational Overhead of Locality Reduction in Binary Optimization Problems." *Computer Physics Communications* 269: 108102. https://doi.org/10.1016/j.cpc.2021.108102.

Välikangas, T., T. Suomi, C. E. Chandler, et al. 2022. "Benchmarking Tools for Detecting Longitudinal Differential Expression in Proteomics Data Allows Establishing a Robust Reproducibility Optimization Regression Approach." *Nature Communications* 13: 7877. https://doi.org/10.1038/s41467-022-35564-z.

van Thieu, N. 2024. "Opfunu: An Open-Source Python Library for Optimization Benchmark Functions." *Journal of Open Research Software* 12. https://doi.org/10.5334/JORS.508.

Van Thieu, N., and S. Mirjalili. 2023. "MEALPY: An Open-Source Library for Latest Meta-Heuristic Algorithms in Python." *Journal of Systems Architecture* 139: 102871. https://doi.org/10.1016/j.sysarc.2023.102871.

Velasco, L., H. Guerrero, and A. Hospitaler. 2024. "A Literature Review and Critical Analysis of Metaheuristics Recently Developed." *Archives of Computational Methods in Engineering* 31: 125–146. https://doi.org/10.1007/S11831-023-09975-0/TABLES/9.

Venkata Rao, R. 2016. "Jaya: A Simple and New Optimization Algorithm for Solving Constrained and Unconstrained Optimization Problems." *International Journal of Industrial Engineering Computations* 7, no. 1: 19–34. https://doi.org/10.5267/j.ijiec.2015.8.004.

Venter, G. 2010. "Review of Optimization Techniques." In *Encyclopedia of Aerospace Engineering*. Wiley. https://doi.org/10.1002/9780470686652.EAE495.

Vermetten, D., C. Doerr, H. Wang, A. V. Kononova, and T. Bäck. 2024. "Large-Scale Benchmarking of Metaphor-Based Optimization Heuristics." In *Proceedings of the Genetic and Evolutionary Computation Conference*, 41–49. ACM. https://doi.org/10.1145/3638529.3654122.

Vigo-Aguiar, J., and P. Alonso. 2013. "New Optimization Techniques in Engineering." *Journal of Mathematical Modelling and Algorithms* 12, no. 3: 213. https://doi.org/10.1007/s10852-013-9223-y.

Vogt, W. 2015. "Tukey's Honestly Significant Difference (HSD) Test." Dictionary of Statistics & Methodology. https://doi.org/10.4135/9781412983907.n2011.

Wang, C. C., C. Y. Ho, C. H. Tu, and S. H. Hung. 2022. "cuPSO: GPU Parallelization for Particle Swarm Optimization Algorithms, in: Proceedings of the ACM Symposium on Applied Computing. https://doi.org/10.1145/3477314.3507142.

Wang, M., B. Li, G. Zhang, and X. Yao. 2018. "Population Evolvability: Dynamic Fitness Landscape Analysis for Population-Based Metaheuristic Algorithms." *IEEE Transactions on Evolutionary Computation* 22: 550–563. https://doi.org/10.1109/TEVC.2017.2744324.

Wang, X., G. G. Wang, B. Song, P. Wang, and Y. Wang. 2019. "A Novel Evolutionary Sampling Assisted Optimization Method for High-Dimensional Expensive Problems." *IEEE Transactions on Evolutionary Computation* 23, no. 5: 815–827. https://doi.org/10.1109/TEVC.2019.2890818.

Wang, Y. 2014. "The Hybrid Genetic Algorithm With Two Local Optimization Strategies for Traveling Salesman Problem." *Computers and Industrial Engineering* 70: 124–133. https://doi.org/10.1016/j.cie.2014.01.015.

Wei, C., Y. Li, and Y. Yu. 2020. "Solution of Ackley Function Based on Particle Swarm Optimization Algorithm." In *Proceedings of the IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA)*. IEEE. https://doi.org/10.1109/AEECA49918.2020.9213634.

Weise, T., R. Chiong, J. Lassig, et al. 2014. "Benchmarking Optimization Algorithms: An Open Source Framework for the Traveling Salesman Problem." *IEEE Computational Intelligence Magazine* 9, no. 3: 40–52. https://doi.org/10.1109/MCI.2014.2326101.

White, D. R., J. McDermott, M. Castelli, et al. 2013. "Better GP Benchmarks: Community Survey Results and Proposals." *Genetic Programming and Evolvable Machines* 14: 3–29. https://doi.org/10.1007/s10710-012-9177-2.

Xiang, X., Y. Tian, R. Cheng, X. Zhang, S. Yang, and Y. Jin. 2022. "A Benchmark Generator for Online Dynamic Single-Objective and Multi-Objective Optimization Problems." *Information Sciences* 613: 591–608. https://doi.org/10.1016/j.ins.2022.09.049.

Xie, L., T. Han, H. Zhou, Z. R. Zhang, B. Han, and A. Tang. 2021. "Tuna Swarm Optimization: A Novel Swarm-Based Metaheuristic Algorithm for Global Optimization." *Computational Intelligence and Neuroscience* 2021: 9210050. https://doi.org/10.1155/2021/9210050.

Yalaoui, F., L. Amodeo, and E.-G. Talbi. 2021. *Heuristics for Optimization and Learning*. Springer.

Yang, L., and A. Shami. 2020. "On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice." *Neurocomputing* 415: 295–316. https://doi.org/10.1016/j.neucom.2020.07.061.

Yang, P., K. Tang, and X. Yao. 2018. "Turning High-Dimensional Optimization Into Computationally Expensive Optimization." *IEEE Transactions on Evolutionary Computation* 22: 143–156. https://doi.org/10.1109/TEVC.2017.2672689.

Yang, X. S. 2010. *Engineering Optimization: An Introduction With Metaheuristic Applications*. John Wiley & Sons. https://doi.org/10.1002/9780470640425.

Yang, X. S., S. Deb, M. Loomes, and M. Karamanoglu. 2013. "A Framework for Self-Tuning Optimization Algorithm." *Neural Computing and Applications* 23: 2051–2057. https://doi.org/10.1007/s00521-013-1498-4.

Yazdani, D., R. Cheng, D. Yazdani, J. Branke, Y. Jin, and X. Yao. 2021a. "A Survey of Evolutionary Continuous Dynamic Optimization Over Two Decades-Part B." *IEEE Transactions on Evolutionary Computation* 25, no. 4: 630–650. https://doi.org/10.1109/TEVC.2021.3060012.

Yazdani, D., R. Cheng, D. Yazdani, J. Branke, Y. Jin, and X. Yao. 2021b. "A Survey of Evolutionary Continuous Dynamic Optimization Over Two Decades-Part A." *IEEE Transactions on Evolutionary Computation* 25, no. 4: 609–629. https://doi.org/10.1109/TEVC.2021.3060014.

Yazdani, D., B. Nouhi, S. Talatahari, D. Yazdani, and A. H. Gandomi. 2023. "Commonly Used Static and Dynamic Single-Objective Optimization Benchmark Problems." In *Handbook of Formal Optimization*. Springer. https://doi.org/10.1007/978-981-19-8851-6_3-1.

Yazdani, D., M. N. Omidvar, R. Cheng, J. Branke, T. T. Nguyen, and X. Yao. 2022. "Benchmarking Continuous Dynamic Optimization: Survey and Generalized Test Suite." *IEEE Transactions on Cybernetics* 52: 3380–3393. https://doi.org/10.1109/TCYB.2020.3011828.

Yazdani, D., M. N. Omidvar, D. Yazdani, K. Deb, and A. H. Gandomi. 2023. "GNBG: A Generalized and Configurable Benchmark Generator for Continuous Numerical Optimization." *arXiv preprint arXiv:2312.07083*. https://arxiv.org/abs/2312.07083.

Zhang, X., H. Liu, X. Wang, L. Dong, Q. Wu, and R. Mohan. 2004. "Speed and Convergence Properties of Gradient Algorithms for Optimization of IMRT." *Medical Physics* 31: 1141–1152. https://doi.org/10.1118/1.1688214.

Zhou, E., and J. Hu. 2014. "Gradient-Based Adaptive Stochastic Search for Non-Differentiable Optimization." *IEEE Transactions on Automatic Control* 59: 1818–1832. https://doi.org/10.1109/TAC.2014.2310052.

Zhou, R., J. Bacardit, A. Brownlee, et al. 2024. "Evolutionary Computation and Explainable AI: A Roadmap to Transparent Intelligent Systems." *IEEE Transactions on Evolutionary Computation*. http://dspace.stir.ac.uk/handle/1893/36283.

Zou, F., D. Chen, H. Liu, S. Cao, X. Ji, and Y. Zhang. 2022. "A Survey of Fitness Landscape Analysis for Optimization." *Neurocomputing* 503: 129–139. https://doi.org/10.1016/j.neucom.2022.06.084.