mathematics

MDPI

Article

Selective Multistart Optimization Based on Adaptive Latin Hypercube Sampling and Interval Enclosures

Ioannis A. Nikas [1,*], Vasileios P. Georgopoulos [2] and Vasileios C. Loukopoulos [2]

[1] Department of Tourism Management, University of Patras, GR 26334 Patras, Greece
[2] Department of Physics, University of Patras, GR 26504 Rion, Greece; vasileios.georgopoulos@upnet.gr (V.P.G.); vxloukop@upatras.gr (V.C.L.)
* Correspondence: nikas@upatras.gr; Tel.: +30-2610962890

Abstract: Solving global optimization problems is a significant challenge, particularly in high-dimensional spaces. This paper proposes a selective multistart optimization framework that employs a modified Latin Hypercube Sampling (LHS) technique to maintain a constant search space coverage rate, alongside Interval Arithmetic (IA) to prioritize sampling points. The proposed methodology addresses key limitations of conventional multistart methods, such as the exponential decline in space coverage with increasing dimensionality. It prioritizes sampling points by leveraging the hypercubes generated through LHS and their corresponding interval enclosures, guiding the optimization process toward regions more likely to contain the global minimum. Unlike conventional multistart methods, which assume uniform sampling without quantifying spatial coverage, the proposed approach constructs interval enclosures around each sample point, enabling explicit estimation and control of the explored search space. Numerical experiments on well-known benchmark functions demonstrate improvements in space coverage efficiency and enhanced local/global minimum identification. The proposed framework offers a promising approach for large-scale optimization problems frequently encountered in machine learning, artificial intelligence, and data-intensive domains.

Keywords: global optimization; multistart; latin hypercube sampling; interval enclosures

MSC: 90C26; 65K05; 65G20; 65G30; 90C56; 68T20

check for updates

Academic Editor: Francesco Alesiani

Received: 5 April 2025
Revised: 20 May 2025
Accepted: 22 May 2025
Published: 24 May 2025

Citation: Nikas, I.A.; Georgopoulos, V.P.; Loukopoulos, V.C. Selective Multistart Optimization Based on Adaptive Latin Hypercube Sampling and Interval Enclosures. *Mathematics* **2025**, *13*, 1733. https://doi.org/10.3390/math13111733

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

1.1. Optimization

Optimization is widely used across various fields, including science, economics, engineering, and industry [1]. An optimization problem aims to find the optimal solution from a set of candidates by minimizing (or maximizing) an objective function [2]. The solution may also be subject to constraints. This work focuses on the box-constrained global minimization problem, formulated as follows:

$$\min_{\mathbf{x}\in\mathcal{S}} f(\mathbf{x}), \tag{1}$$

where f is the objective function, $\mathcal{S}$ is a closed hyper-rectangular region of $\mathbb{R}^n$, and the vector $\mathbf{x} = (x_1, \ldots, x_n)$ is the optimization variable. A vector $\mathbf{x}^\star$ is the optimal solution, or the *global minimum*, of problem (1),] if it attains the lowest objective value among all vectors in $\mathcal{S}$, i.e.,

$$f(\mathbf{x}^\star) \leq f(\mathbf{x}), \quad \forall \mathbf{x} \in \mathcal{S}. \tag{2}$$

A vector $\mathbf{x}^\star$ is a *local minimum* if there exists a neighborhood $\mathcal{N} \subset \mathcal{S}$ such that $f(\mathbf{x}^\star) < f(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{N}$, with $\mathbf{x} \neq \mathbf{x}^\star$.

1.2. Optimization in the Modern Era

In recent years, Artificial Intelligence (AI), Machine Learning (ML), Deep Learning (DL), and Big Data (BD) have revolutionized nearly all areas of science [3,4], becoming a key driver of the Fourth Industrial Revolution [5]. Optimization plays a fundamental role in all these fields: machine learning algorithms use data to formulate an optimization model, whose parameters are determined by minimizing a loss function [6]. Deep learning follows the same principle; however, optimization in deep networks is more challenging due to their complex architectures and the large datasets involved [7]. Meanwhile, data scale is also a vital factor. Big data is commonly characterized by the "5 Vs": volume, variety, value, velocity, and veracity [5]. Managing these factors effectively is essential for insight discovery, informed decision-making, and process optimization [8].

Thus, optimization is the backbone of AI, ML, DL, and BD, enabling models to learn efficiently, make better decisions, and handle large-scale computations [6]. From training AI models and tuning hyperparameters to designing deep learning architectures and managing big data workflows, optimization is crucial for achieving high performance and efficiency. Consequently, efficient optimization is more crucial than ever.

1.3. Local and Global Optimizers

Optimization algorithms are broadly classified into local and global ones. Global optimizers explore large regions of the search space to locate the global optimum [9]. Global optimization is essential when the objective function contains multiple local optima. In contrast, local optimizers iteratively refine an initial solution within a confined search region, seeking a local optimum. Local optimization is highly sensitive to initial conditions, as different starting points may lead to different local optima [2]. This approach is suitable when the global optimum is expected within a specific region of the search space.

Gradient-based local optimizers include gradient descent, Newton's method, and quasi-Newton methods. Gradient descent updates the solution by following the negative gradient, ensuring monotonic improvement under convexity conditions [1]. Newton's method leverages second-order derivatives for faster convergence but requires computationally expensive Hessian evaluations [10]. Quasi-Newton methods, such as the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm, approximate the Hessian efficiently, balancing computational cost and convergence speed [11].

Derivative-free local optimizers are useful when gradient computation is impractical or costly. Methods such as Nelder–Mead and Pattern Search refine solutions heuristically, avoiding the need for explicit derivative calculations [12,13]. These techniques are widely applied to non-differentiable or noisy objective functions.

Global optimization methods fall into two main categories: deterministic and stochastic, depending on whether they incorporate stochastic elements [14,15]. Deterministic algorithms systematically explore the feasible space, whereas stochastic methods rely on random sampling, with probabilistic convergence as their defining characteristic [16]. Among deterministic methods, interval techniques are among the most widely used [17–19]. These techniques iteratively partition the search space $\mathcal{S}$ into smaller subregions, discarding those that do not contain the global solution with certainty, using the principles of interval analysis. However, most global optimization methods are stochastic, including controlled random search (CRS), simulated annealing (SA), differential evolution (DE), genetic algorithms (GAs), particle swarm optimization (PSO), and ant colony optimization (ACO), among others [20].

In addition, there are hybrid stochastic methods which combine different optimization techniques to enhance performance. Examples include PSO–SA hybrids, GA–DE hybrids, and GA–PSO hybrids [20].

1.4. The Multistart Method

The multistart method (MS) [21,22] is one of the main stochastic global optimization techniques and serves as the foundation for many modern global optimization methods. The algorithm aims to efficiently explore the feasible space by ensuring that each region is searched only once [16]. Several variants have been developed, including clustering [23], domain elimination [24], zooming [24], and repulsion [25]. These variants improve the method by introducing new features or addressing its weaknesses, such as repeatedly identifying the same minimum [16].

Typically, MS operates by capturing a random sample from the objective function, which is then used as a starting point for a local optimizer. Samples are typically drawn from a specified probability distribution, most commonly the uniform distribution. This approach is particularly useful for non-convex problems with multiple local optima, as it increases the likelihood of locating a global optimum. The multistart method has been extensively studied in various contexts, including its application to identifying local minima [26], its integration into hybrid techniques [27], and its use in parallel optimization [28].

The term *selective* multistart optimization describes a multistart framework in which local optimization is not performed from all available sampling points but only from a selected subset identified as the most promising. Similar approaches have been proposed in the literature, aiming to improve the efficiency of multistart methods through intelligent sampling, ranking, or filtering strategies based on probabilistically "best" points in their neighborhood [29].

1.5. Sampling

The distribution of starting points significantly impacts the performance of a multistart method. Several sampling strategies exist, each with its own advantages and drawbacks.

Random sampling is the simplest approach, selecting points uniformly at random from the search space. While easy to implement, it may lead to inefficiencies, especially in high-dimensional spaces where random points may be sparsely distributed [30].

Quasi-random sequences, such as Halton and Sobol sequences, provide low-discrepancy sampling, ensuring better uniformity than purely random sampling. This makes them more effective for multistart optimization [31].

Adaptive sampling methods use information from previous function evaluations to guide future sampling. Techniques such as clustering-based sampling and Bayesian Optimization (BO) dynamically adjust sampling density in regions of interest, improving convergence rates [32].

Latin Hypercube Sampling enhances random sampling by ensuring a more uniform coverage of the search space. Each variable's range is divided into equal intervals, with points selected so that each interval is sampled exactly once. This method reduces clustering and ensures better space-filling properties [33]. Various modifications of Latin Hypercube Sampling have been proposed to address different types of problems [34–36].

1.6. Current Work

This paper proposes a novel framework for selective multistart optimization to identify both local and global minima. The framework enhances the effectiveness and efficiency of multistart optimization by addressing key limitations through the integration of Latin Hypercube Sampling and Interval Arithmetic. Its primary objectives are to improve space coverage of the feasible search space, particularly in higher dimensions; prioritize

sampling points from less to more promising ones, to guide optimization toward the global minimum; ensure uniform behavior regardless of the resulting sample; and maintain computational efficiency.

LHS provides uniformity, while IA introduces two key advantages: (a) it enables the discard of regions that do not contain the global minimum with certainty, thereby reducing the search space, and (b) it enables the verification of a minimum as the global minimum—a capability not available using Real Arithmetic. These features enhance both effectiveness and efficiency, particularly in higher dimensions. The proposed framework's performance is compared to that of a conventional multistart approach using the same sample. Rather than selecting points randomly within the interval boxes, this work selects the midpoint.

Both approaches are implemented in MATLAB (v. 24.2.0.2773142), using a combination of core MATLAB functions, proprietary MathWorks toolbox functions—namely, `lhsdesign` (for Latin Hypercube Sampling), from the Statistics and Machine Learning Toolbox, and `fminunc` (a quasi-Newton method), from the Optimization Toolbox—as well as functions from the third-party INTLAB toolbox for interval arithmetic [37,38]. All experiments were conducted on an AMD Ryzen 2700X, equipped with 32 GB of RAM, on Windows 10.

2. Materials and Methods

2.1. LHS and the Coverage Issue

Stratified sampling is a variance reduction technique in which the sample space Ω is partitioned into M disjoint strata $\{D_m\}$, where $m = 1, 2, \ldots, M$, for some $M \geq 2$, and a sample is drawn independently from each stratum. These strata satisfy $D_i \cap D_j = \varnothing$ for all $i \neq j$, ensuring that their union covers the entire space, i.e., $\Omega = \bigcup D_m$.

A specific form of stratified sampling is Latin Hypercube Sampling, which further subdivides each of the d dimensions of the domain D_k into N non-overlapping (except at their endpoints), equally sized intervals $\{D_{k,j}\}$, $j = 1, 2, \ldots, N$. Each interval $D_{k,j}$ contains exactly one sampled point, drawn uniformly at random within that interval. LHS achieves a more uniform distribution of points across the domain, thereby reducing sampling variability. For each dimension $k \in \{1, \ldots, d\}$, we define the partition $D_k = \bigcup_j D_{k,j}$ with $D_{k,i} \cap D_{k,j} = \varnothing$ whenever $i \neq j$, $i, j = 1, 2, \ldots, N$. LHS is performed using the density function $f_k(x)\mathbb{1}(x \in D_{k,j})/N$, where f_k represents the marginal probability density function of the kth variable. The final random vector $\boldsymbol{x}$ is then formed by permuting the sampled values across dimensions (Figure 1) [39].

The percentage coverage of the search space can be approximated using d-dimensional intervals or interval boxes, from which the sample is drawn. A partition of N non-overlapping intervals in each dimension leads to the following formula for search space coverage:

$$c(N, d) = \frac{N}{N^d} \cdot 100 = \frac{100}{N^{d-1}}, \tag{3}$$

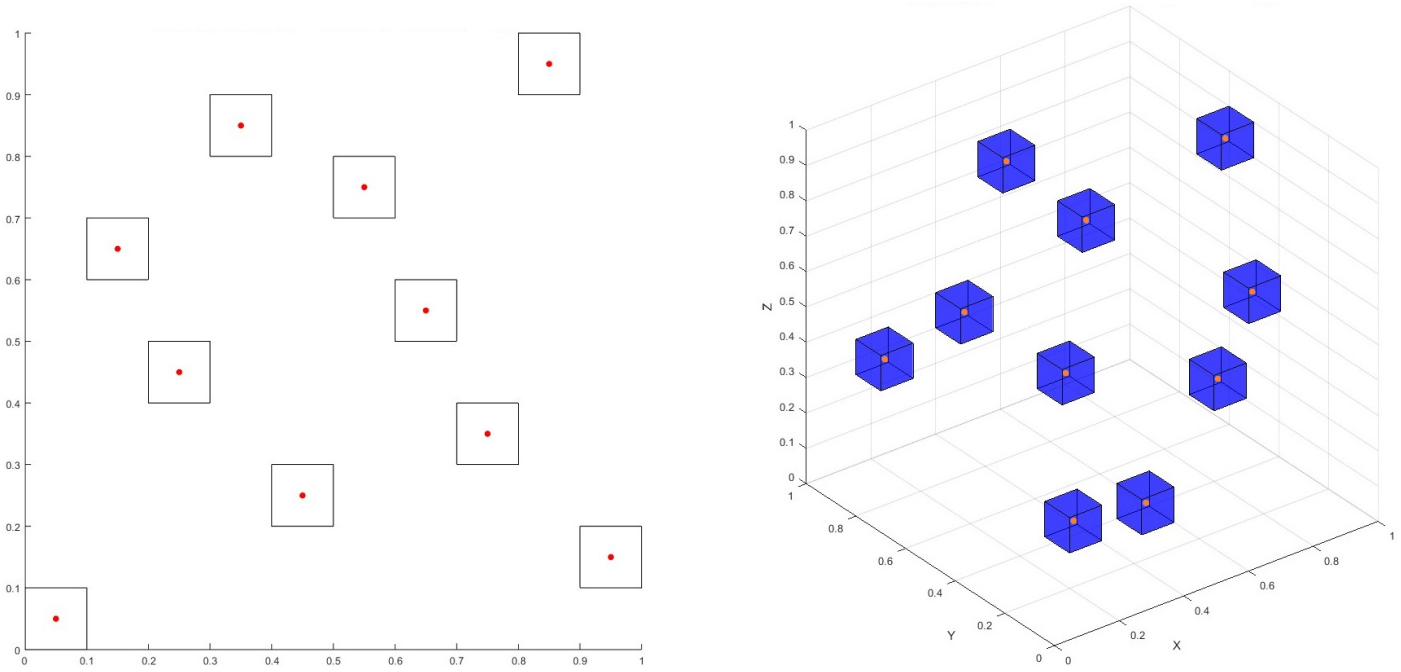
where c denotes the percentage coverage of the search space. It is evident that as the number of partitions or the dimensionality increases, the coverage c tends to zero. Conversely, to maintain a minimum coverage of the search space, say $c\%$, the required partitioning in each dimension is given by:

$$N = \left\lceil 10^{\frac{2-\log c\%}{d-1}} \right\rceil. \tag{4}$$

This formula leads to an interesting conclusion:

Corollary 1 (The Coverage Issue)**.** *The percentage coverage of the search space using interval boxes resulting from LHS is exponentially inversely proportional to the number of required partitions as the dimensionality of the problem increases.*

In practice, this implies that as the dimensionality of the problem increases, achieving constant coverage of the search space using the LHS requires increasingly finer partitions. In contrast, if we use interval boxes, achieving constant coverage requires fewer and fewer boxes, which, in the context of multistart optimization, leads to fewer applications of a local minimizer and, consequently, to a lower probability of finding the global minimum.



(**a**) (**b**)

Figure 1. Latin Hypercube Sampling: Interval boxes and their corresponding midpoints. In (**a**) two- and (**b**) three-dimensional space.

2.2. Proposed Framework Components

The proposed selective multistart optimization framework consists of the below components.

2.2.1. Latin Hypercube Sampling

The first component of the framework is implemented using LHS (see Section 2.1).

2.2.2. Constant Coverage of Search Space/Resolving the Coverage Issue

In the second component of the framework, in light of Corollary 1, we begin with a given partitioning and coverage rate of the search space. To ensure the required coverage, an appropriate sample size is chosen, which, however, corresponds to a larger partition than the given one:

$$S(c, N, d) = \left\lceil \frac{c}{100} \cdot N^d \right\rceil, \tag{5}$$

where $S \in \mathbb{N}$ is the sampling size function, $c\% \in [0, 1]$ the desired coverage, $N \in \mathbb{N}$ the partition scheme, and $d \in \mathbb{N}$ the problem's dimensionality. To resolve the discrepancy between the given and required partitions, the interval boxes corresponding to the sampling points are expanded to match the size indicated by the given partition.

2.2.3. Priorities on the Sampling Points

In the third component of the framework, the range of the objective function is estimated over the expanded interval boxes using interval arithmetic. These boxes are assessed by the lower bound of their calculated enclosures; those with the lowest lower bounds are deemed most promising for locating the global minimum. This process reflects on the corresponding sample points, determining which initial value is the most promising, guiding the local minimizer toward the global minimum.

Under certain conditions, interval arithmetic may overestimate the range of an objective function [40], providing realistic upper and lower bounds while identifying promising regions for deeper local searches. For example, consider the one-dimensional Rastrigin function, defined over $[-5.12, 5.12]$, as shown in Figure 2. The given partition size is 6, and for each partition, a range estimation is performed using interval arithmetic on the natural interval extension of the Rastrigin function—a *natural interval extension* replaces the real variables of the objective function with interval ones. It is evident that the third and fourth intervals are the most likely to contain the global minimum. Therefore, the points drawn within these intervals using LHS should be prioritized to initiate a local minimizer.

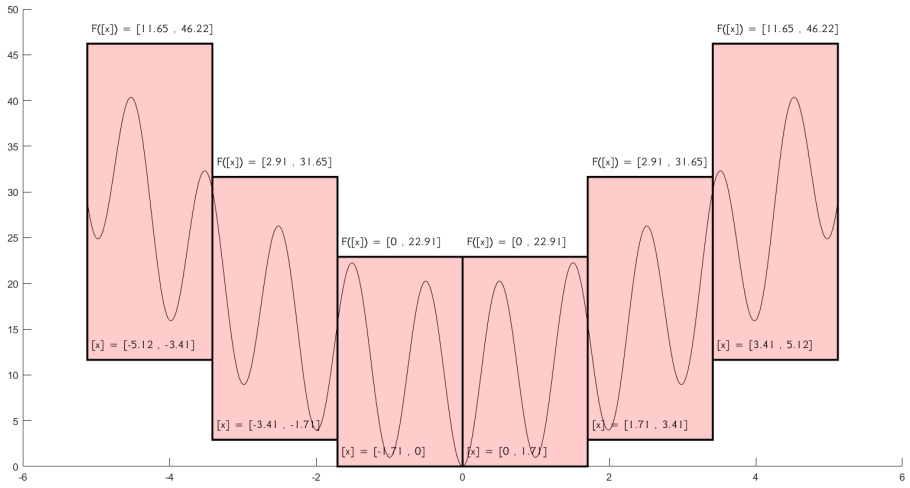


Figure 2. One−dimensional Rastrigin function: Range overestimation on six-box partition. The two middle boxes are identified as the most promising for deeper local search.

Range estimation for each interval box enables local comparison of the objective function's dynamics and, in some cases, the discard of certain search space regions from further exploration. Specifically, in interval global optimization, pruning techniques such as the cut-off test [41] can accelerate convergence by discarding interval boxes that are guaranteed not to contain the global minimum. During the cut-off test, an interval box is eliminated from further consideration if its lower bound estimate exceeds a known upper bound ($\tilde{f}$) identified elsewhere in the search space (Figure 3).

2.2.4. Multistart, "Promising Status", and Termination Criteria

The search continues until the local minimizer has processed all available sample points or predefined termination criteria are met. In this paper, two termination criteria are used: (1) a criterion to accept a local minimum as the global minimum with certainty, and (2) a criterion to accept a successively found local minimum as a candidate for the global minimum [42].

Termination Criterion 1 (Strong Condition)**.** *Let f be an objective function, $[x] \in \mathbb{IR}$ an interval box, and $\tilde{f}$ a known upper bound (i.e., a previously found local minimum) of the global minimum $f^\star$. Let F be the natural interval extension of f and $F([x])$ be the range enclosure of f*

over $[x]$. *If* $|\tilde{f} - \underline{F}([x])| < \varepsilon$ *for a small predefined tolerance* ε, *then the global minimum has been found with tolerance* ε.

Termination Criterion 2 (Weak Condition). *Let* f *be an objective function, and* x^k_{local} *and* x^{k+1}_{local} *successive local minimizers found by the multistart optimization method. If* $|x^{k+1}_{local} - x^k_{local}| < \varepsilon$ *for a small predefined tolerance* ε, *then* $f(x^{k+1}_{local})$ *is considered a candidate global minimum with tolerance* ε.

Each time a local minimum is found, the "promising status" of the available sample points is reassessed using well-known acceleration devices in interval global optimization, such as the cut-off test, applied to the corresponding expanded interval boxes. If applicable, these techniques discard certain interval boxes, reducing the number of sample points that need to initiate a local search.

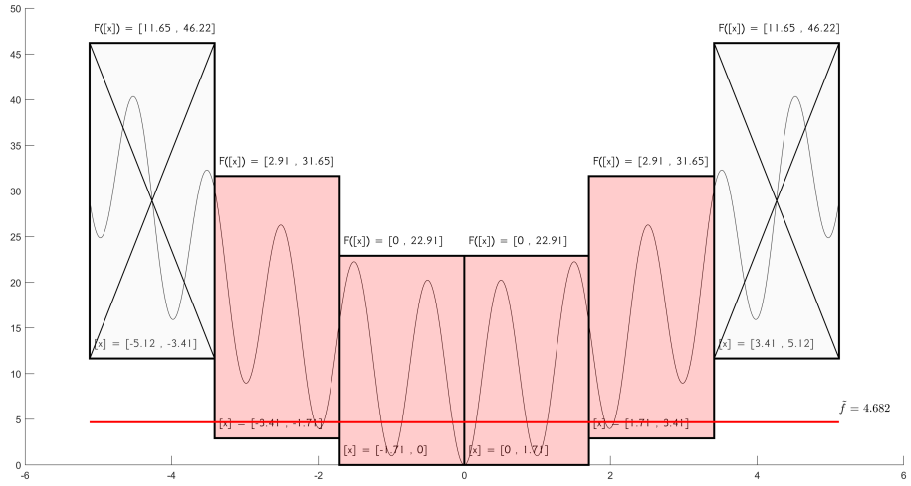


Figure 3. One−dimensional Rastrigin function: Application of the cut-off test results in the exclusion of 2 out of 6 interval boxes.

2.3. On Interval Analysis

A closed, compact interval, denoted as $[x]$, is the set of all points

$$[x] = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} \ \ \underline{x} \leq x \leq \overline{x}\}$$

where $\underline{x}$, $\overline{x}$ denote the lower and upper bound of interval $[x]$, respectively, and the set of all closed and compact intervals is denoted as $\mathbb{IR}$. The intersection of two intervals $[x] \cap [y]$ is defined as the common points between the two intervals, while the empty intersection, the lack of common points is defined as $[/]$. The width of an interval $[x]$ is defined by $w([x]) = \overline{x} - \underline{x}$, while the midpoint of an interval $[x]$ is defined as $m([x]) = \frac{\overline{x}+\underline{x}}{2}$.

Similarly, an interval vector $[\boldsymbol{x}]$, also called an interval box, is a vector of intervals $[\boldsymbol{x}] = ([x]_1, [x]_2, \ldots, [x]_n)^T$, and the set of all interval vectors is denoted as $\mathbb{IR}^n$. The width of an interval vector is defined by

$$w([\boldsymbol{x}]) = \max_i \{w([x]_i)\},$$

while the midpoint of an interval vector is defined as

$$m([\boldsymbol{x}]) = \{m([x]_1), m([x]_2), \ldots, [x]_n\}.$$

For operations between two intervals $[a], [b] \in \mathbb{IR}$, a corresponding interval arithmetic was defined, which, in general, is described as a set extension of real arithmetic, using the elementary operations $\circ \in \{+, -, \cdot, \div\}$

$$[a] \circ [b] = \{a \circ b \ \ \forall a \in [a], \forall b \in [b]\}.$$

The above definition is equivalent to the following simpler rules [43],

$$\begin{aligned}
[a] + [b] &= [\underline{a} + \underline{b}, \overline{a} + \overline{b}] \\
[a] - [b] &= [\underline{a} - \overline{b}, \overline{a} - \underline{b}] \\
[a] \cdot [b] &= \left[\min\left\{\underline{ab}, \underline{a}\overline{b}, \overline{a}\underline{b}, \overline{ab}\right\}, \max\left\{\underline{ab}, \underline{a}\overline{b}, \overline{a}\underline{b}, \overline{ab}\right\}\right], \\
\frac{[a]}{[b]} &= \left[\min\left\{\frac{\underline{a}}{\underline{b}}, \frac{\underline{a}}{\overline{b}}, \frac{\overline{a}}{\underline{b}}, \frac{\overline{a}}{\overline{b}}\right\}, \max\left\{\frac{\underline{a}}{\underline{b}}, \frac{\underline{a}}{\overline{b}}, \frac{\overline{a}}{\underline{b}}, \frac{\overline{a}}{\overline{b}}\right\}\right]
\end{aligned}$$

where $0 \notin [b]$ for the case of interval division. For the case where $0 \in [b]$, an extended interval arithmetic has been proposed, [44].

The range of a function f over a closed and compact interval vector $[\boldsymbol{x}] \subset \mathbb{R}^n$ is defined as

$$f_{rg}([\boldsymbol{x}]) = \{f(\boldsymbol{x}) \ \ x \in [\boldsymbol{x}]\} \subseteq F([\boldsymbol{x}]),$$

where F is an interval extension of function f and provides enclosures of the actual range of f. The following theorem, known as the Fundamental Theorem of Interval Analysis [40,45], can be used to bound the exact range of a given expression.

Theorem 1. *Let $f : \mathbb{IR}^n \to \mathbb{IR}$ be a function, and let $[\boldsymbol{x}]$ be an interval vector. If F is an inclusion isotonic interval extension of f, then*

$$f([x]) \subseteq F[([x])]. \tag{6}$$

The resulting enclosure of the range f is generally overestimated due to the phenomenon of dependency, as described in [45]. Specifically, the more occurrences a variable has in an expression, the higher the overestimation will be. A more thorough study on interval analysis may be found indicatively in [40,46–48].

2.4. The Algorithmic Approach

All the aforementioned ideas are summarized in Algorithm 1 and Figure 4, which outline the proposed framework for selective multistart optimization (`SelectiveMultistart`).

The proposed framework is designed for global optimization by efficiently exploring the search space. The process begins with the necessary initializations, followed by the calculation of the sample size required to achieve the desired coverage c% and the application of LHS to generate the sample. The interval boxes containing the sample points are then expanded to ensure that their total number matches the given partition. The objective function is then evaluated for all expanded intervals, and the results are stored in a working list $\mathcal{L}$, along with the lower bound of the corresponding range estimation. Steps 1 through 6 constitute the preprocessing phase of the proposed framework.

Algorithm 1: Selective multistart framework for global optimization.

Function ***SelectiveMultistart*** *(F, $[\boldsymbol{x}]_0$, N, d, c, ε_{opt}, ε_{ms})*

$\tilde{f} := +\infty$; $x^k_{local} = +\infty$ // Initialization

$FL = \underline{F}([\boldsymbol{x}]_0)$ // $f^\star \geq \underline{F}([\boldsymbol{x}])$

$S := \left\lceil \frac{c}{100} \cdot N^d \right\rceil$ // Sample Size S

Sample := LHS(d, S);

$[\boldsymbol{x}]$:= Expand_IBoxes($Sample$, N) // We construct the intervals that contain each sample point

$\mathcal{L} := \{([\boldsymbol{x}]_i, \underline{F}([\boldsymbol{x}]_i))\}$;

while $L \neq \{\emptyset\}$ **do**

$[\boldsymbol{y}]$:= Best_Point(L) // The most promising

$[f_{min}, x_{min}]$:= Local_Optimizer($mid(\boldsymbol{y})$, ε_{opt});

$\tilde{f} := \max(f_{min}, \tilde{f})$;

// Termination criterion #1--Strong condition

if $|f_{min} - FL| \leq \varepsilon_{ms}$ **then**

$f^\star := f_{min}$ // Global minimum is found

return;

end

// Termination criterion #2--Weak condition

else if $|x^{k+1}_{local} - x^k_{local}| \leq \varepsilon_{ms}$ **then**

$f^\star := f_{min}$ // Candidate global minimum

return;

end

else

L := CutOffTest(L, $\tilde{f}$) // Discard all boxes that $\underline{F}([\boldsymbol{x}] > \tilde{f}$

$x^k_{local} = x^{k+1}_{local}$

end

end

end

Function ***Expand_IBoxes*** *(S,N)*

$ii = [S \cdot N]$; // Interval Index

$is = w(axis)/N$; // Interval Size

$[x] = [inf(SI) + (ii - 1) \cdot is, inf(SI) + ii \cdot is]$; // Constructed Intervals

return $[x]$;

end

The subsequent steps form the core of the multistart algorithm. The most promising point is selected from the working list, based on the lower bound of its range estimation, and extracted for local optimization. A local minimizer is then initiated, and the best-known upper bound is compared with the newly found local minimum, updating the bound if a better solution is identified. The termination criteria are then applied, followed by the execution of the cut-off test on the remaining items on the working list. The algorithm iterates through this sequence of steps until the termination criteria are met or no promising interval boxes remain.

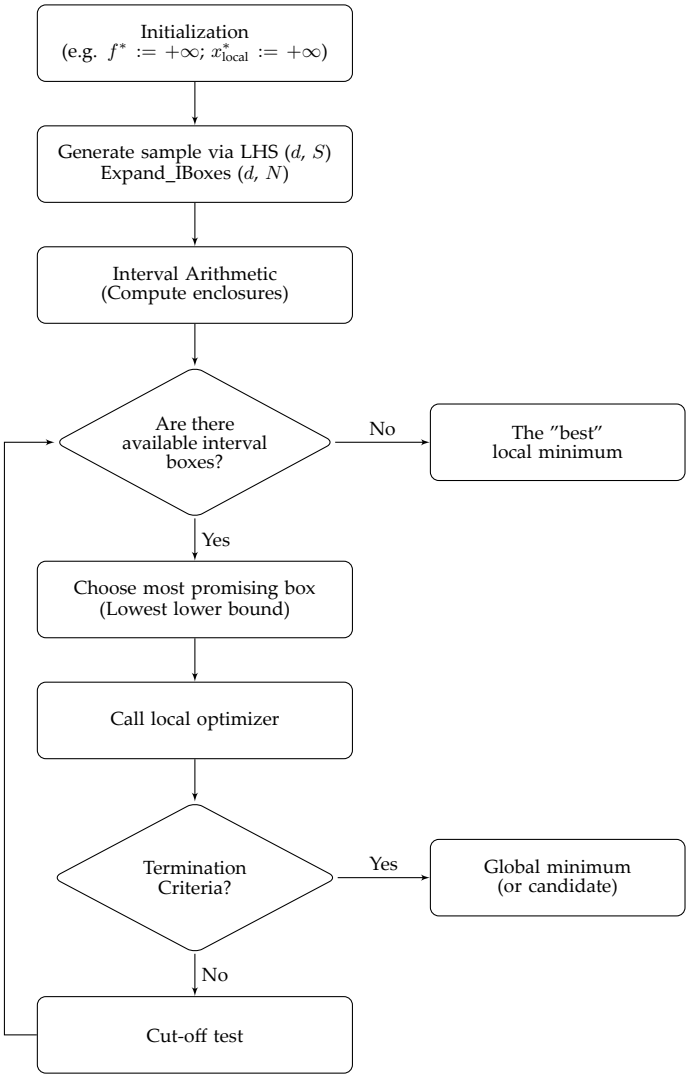


Figure 4. Flowchart of the proposed selective multistart optimization framework. The method combines Latin Hypercube Sampling, interval analysis, and local search with termination and pruning criteria.

3. Results

In the literature, there is no consensus on how to demonstrate a method's efficiency, and a variety of metrics and graphical representations are used. This work presents the following outputs, motivated by the following considerations: (a) the method's effectiveness in detecting the global minimum; (b) in cases where detection fails, the method's proximity to the global minimum; (c) the method's efficiency (i.e., computational cost relative to output); and (d) the method's behavior with respect to the problem's characteristics. Accordingly, Figures 5–9, Tables 1–15 and A1–A5 (in Appendix A) were selected to illustrate the above aspects.

Our proposed framework is referred to as *IVL*, while the conventional approach is denoted as *MS*. All graphs and tables are presented such that a positive value indicates better performance of IVL compared to MS, whereas a negative value indicates worse performance. Entries marked with an asterisk (*) correspond to cases of division by zero. Specifically, in Figures 5–9:

- Graph (a) illustrates the difference in success rates between the two methods, where success rate is defined as the ratio of successfully identified global minima to the total number of function calls.
- Graph (b) presents the correlation between success rate and problem dimensionality for each partition size, providing insights into how scalability affects the detection capability of each method.
- Graph (c) shows the relationship between the number of function evaluations and the partition size, across all dimensions, highlighting the computational cost associated with increasing dimensionality.
- Graph (d) investigates efficiency by correlating the number of function calls per detected local minimum with partition size, again across all dimensions.
- Graph (e) depicts the relative difference in function evaluations between the two methods across all partition sizes and dimensions.
- Finally, Graph (f) introduces an efficiency metric for IVL, defined as:

$$\frac{\text{total number of search interval boxes} - \text{function calls}}{\text{total number of search interval boxes}}$$

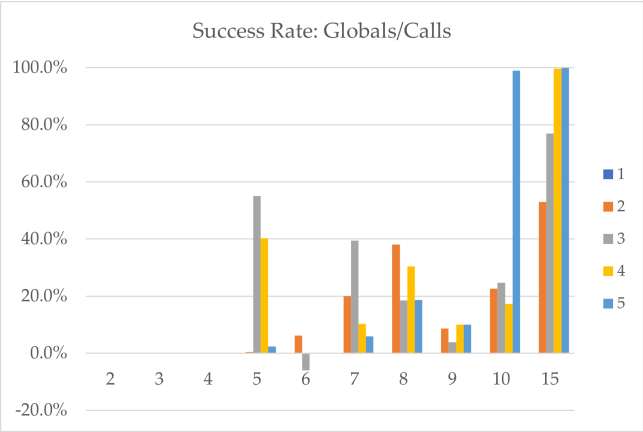
  which quantifies how effectively the method avoids unnecessary evaluations, thereby reflecting its ability to prune the search space.

Each table reports three performance metrics—namely, Success Rate (SR), Convergence Variability (CV), and Efficiency (Eff)—for both IVL and MS. The symbol * denotes division with zero.

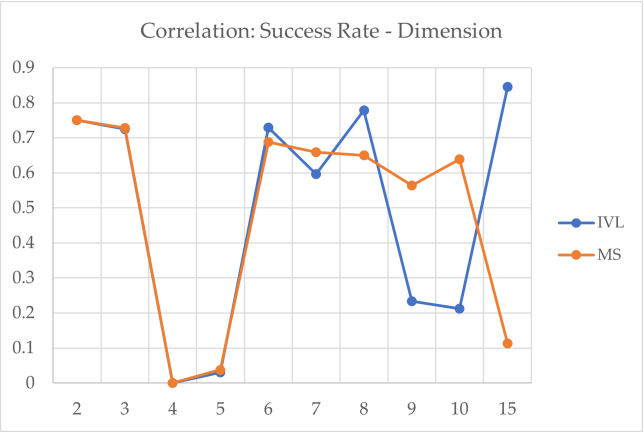
- The Success Rate (SR) is defined as the number of successfully detected global minima divided by the total number of function evaluations, serving as a measure of effectiveness.
- The Coefficient of Variation (CV) is defined as the ratio of the standard deviation to the mean of the distances between the detected minima to the true global minimum. This metric highlights the robustness of convergence across trials.
- The Efficiency (Eff) metric is calculated as the success rate divided by the number of function calls, providing an indicator of performance that balances accuracy with computational cost.

Additional ratios and measures are provided in Tables A1–A5 (in Appendix A).

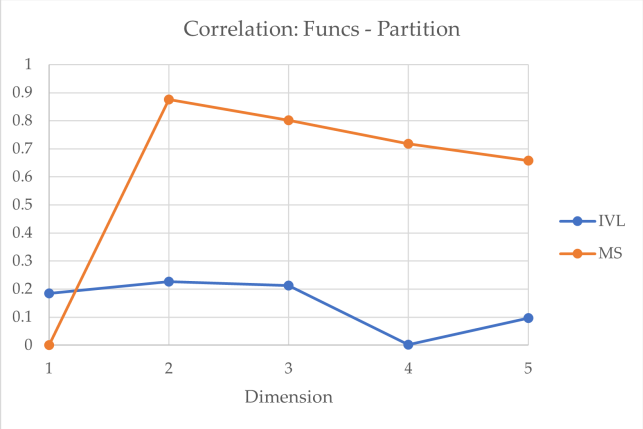
The experiments were conducted using a space coverage of 10%, with a tolerance of $\varepsilon = \varepsilon_{opt} = \varepsilon_{ms} = 1 \times 10^{-6}$. These values reflect a balance between practical numerical accuracy and computational feasibility.



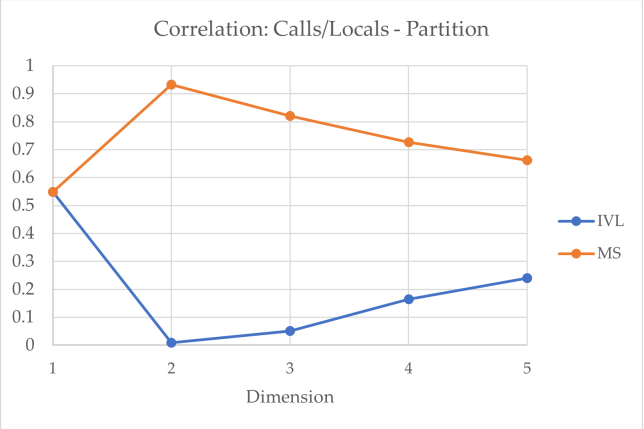
(**a**)



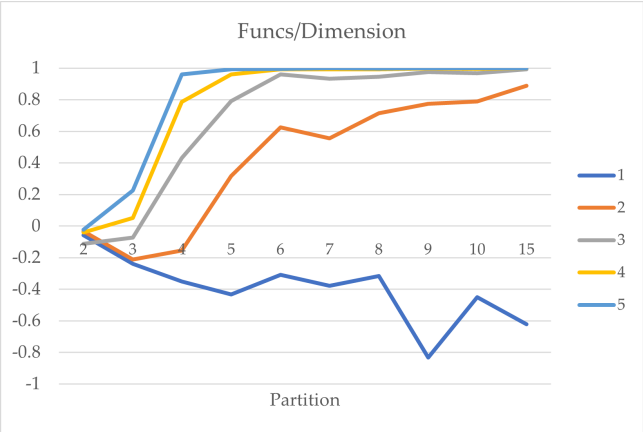
(**b**)



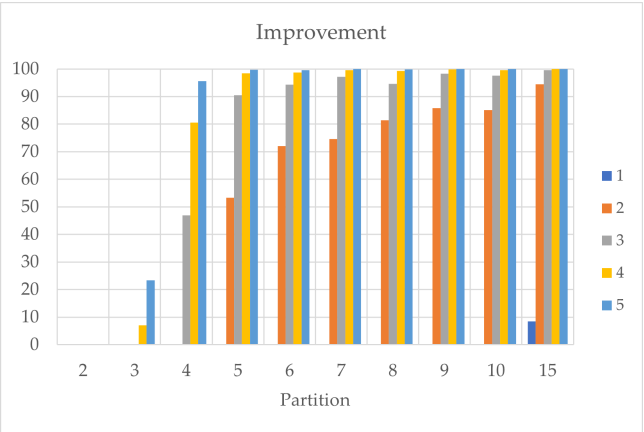
(**c**)



(**d**)

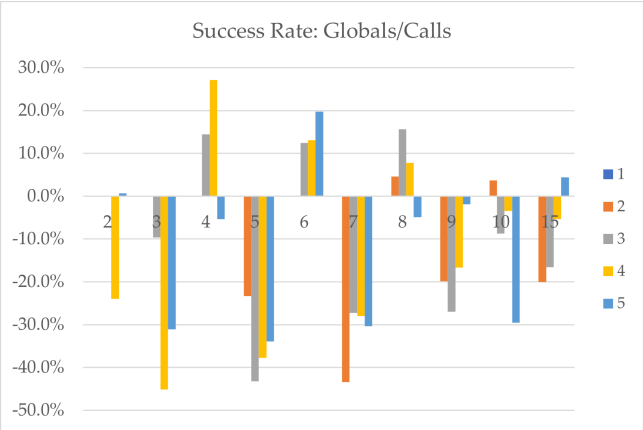


(**e**)

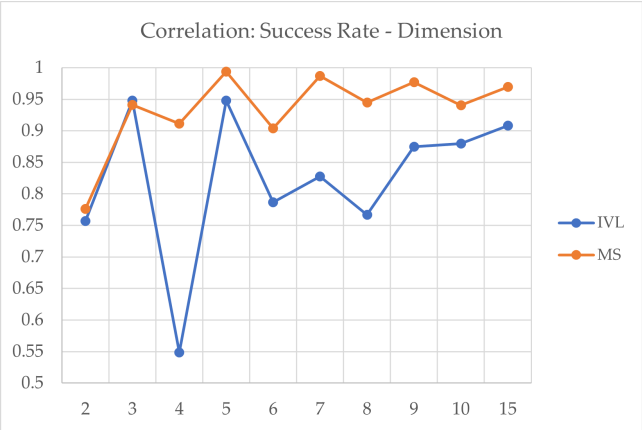


(**f**)

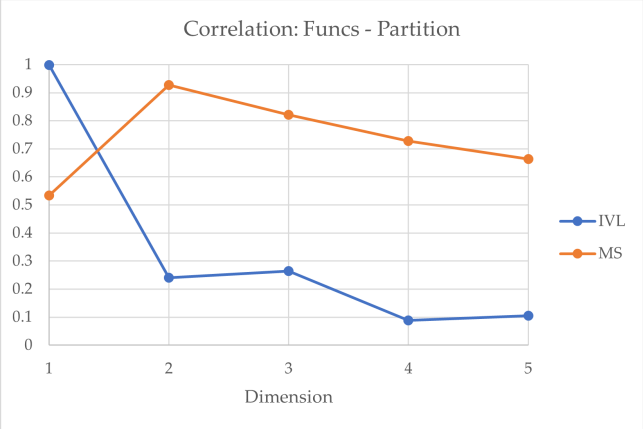
Figure 5. Ackley Function: (**a**) Success Rate, (**b**) Correlation: Success Rate−Partition, (**c**) Correlation: Function Evaluations−Partition, (**d**) Correlation: Calls/Locals−Partition, (**e**) Function Evaluations/Dimension, and (**f**) Improvement.



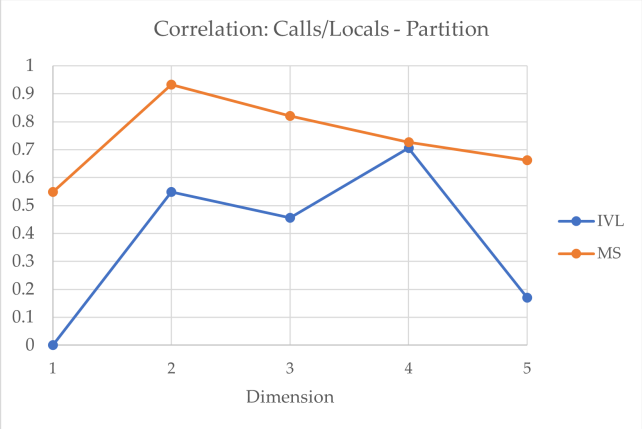
(**a**)



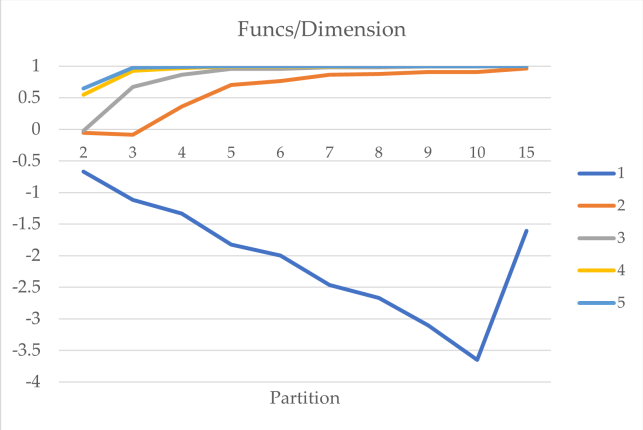
(**b**)



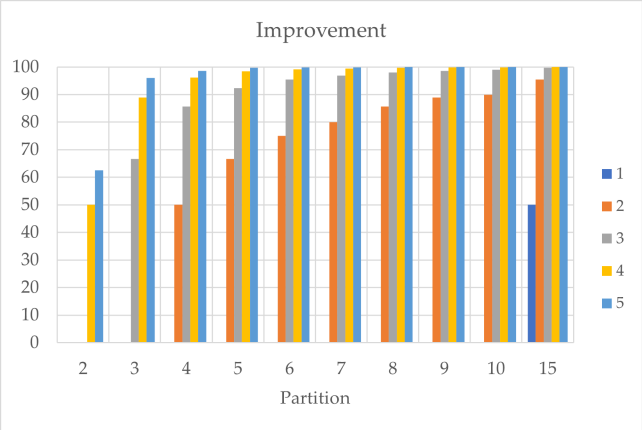
(**c**)



(**d**)

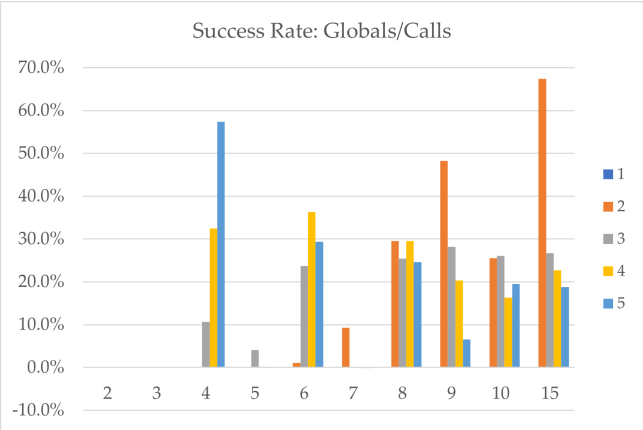


(**e**)

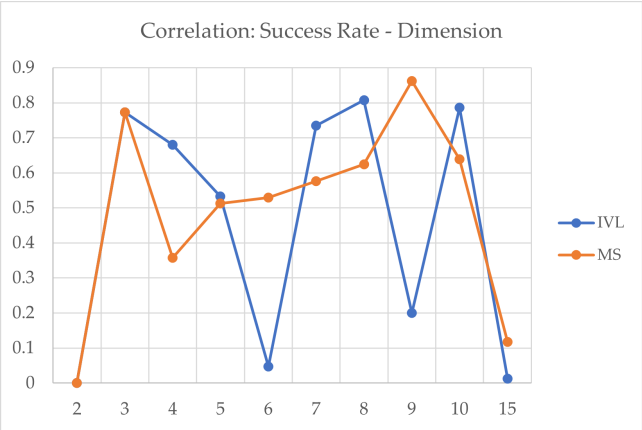


(**f**)

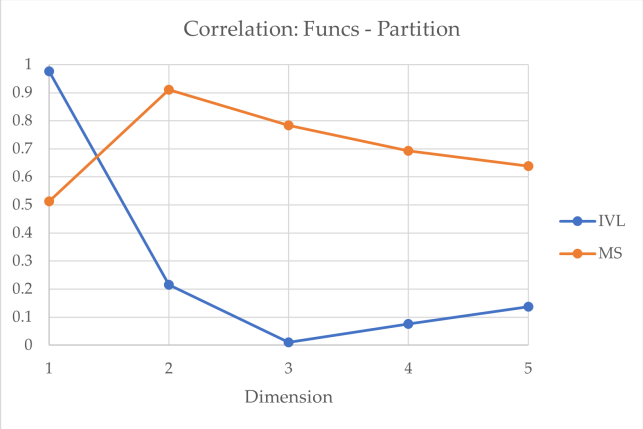
Figure 6. Dixon–Price Function: (**a**) Success Rate, (**b**) Correlation: Success Rate−Partition, (**c**) Correlation: Function Evaluations−Partition, (**d**) Correlation: Calls/Locals−Partition, (**e**) Function Evaluations/Dimension, and (**f**) Improvement.



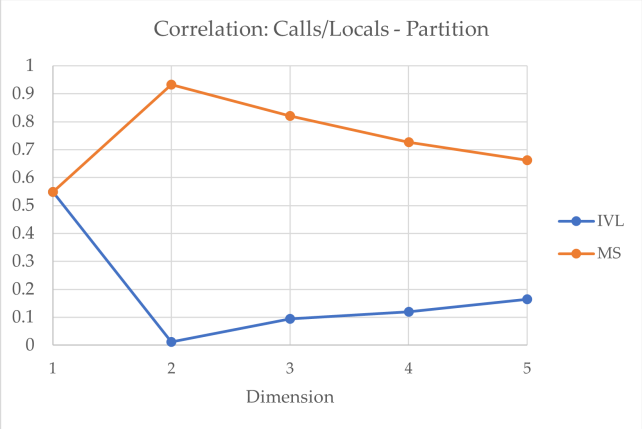
(**a**)



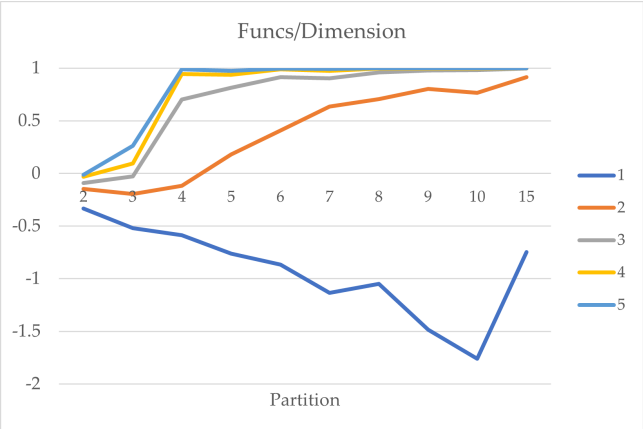
(**b**)



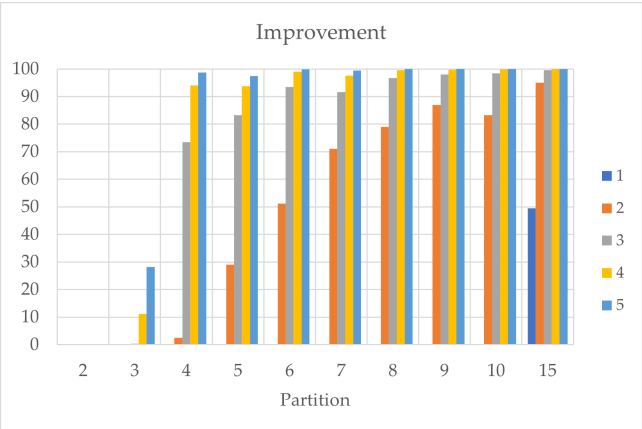
(**c**)



(**d**)

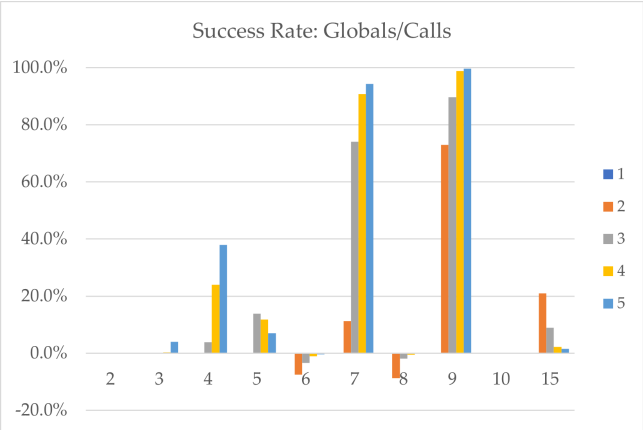


(**e**)

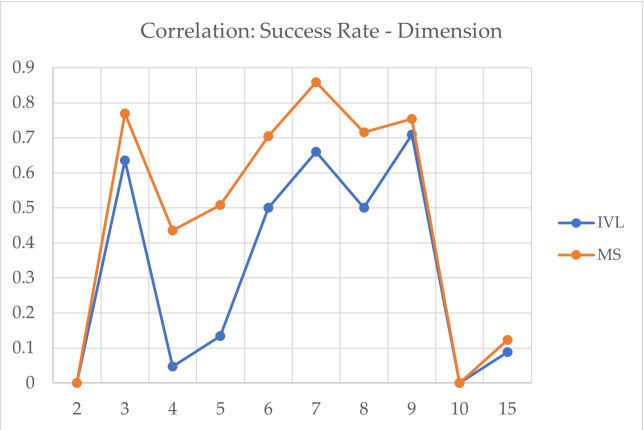


(**f**)

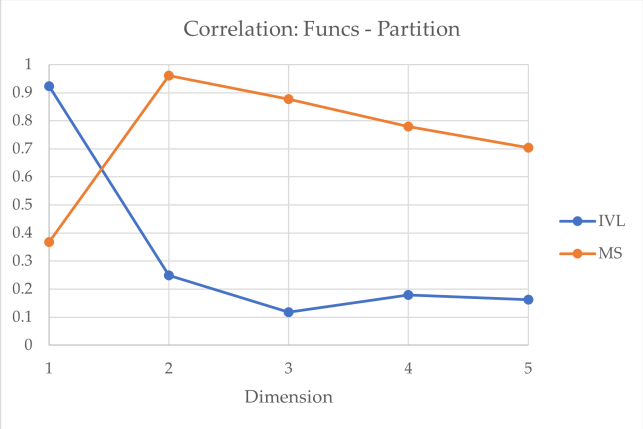
Figure 7. Levy Function: (**a**) Success Rate, (**b**) Correlation: Success Rate—Partition, (**c**) Correlation: Function Evaluations—Partition, (**d**) Correlation: Calls/Locals—Partition, (**e**) Function Evaluations/Dimension, and (**f**) Improvement.



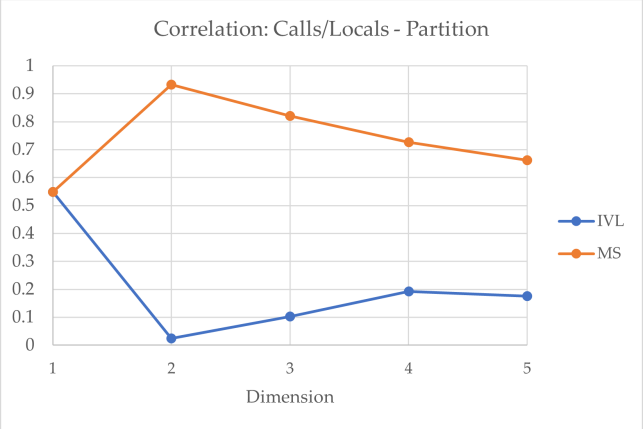
(**a**)



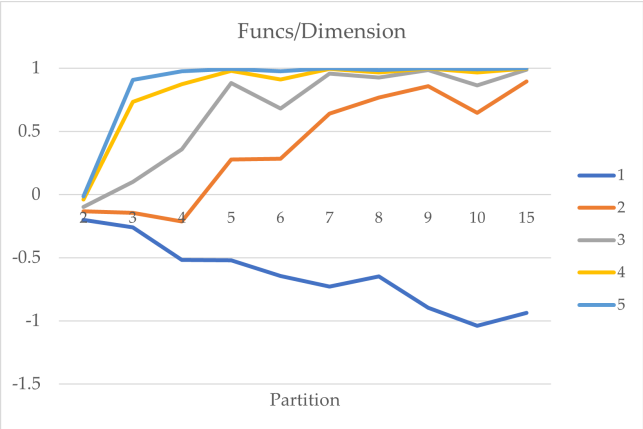
(**b**)



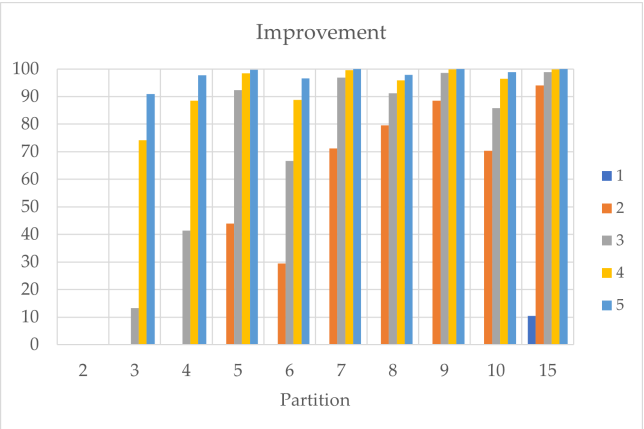
(**c**)



(**d**)

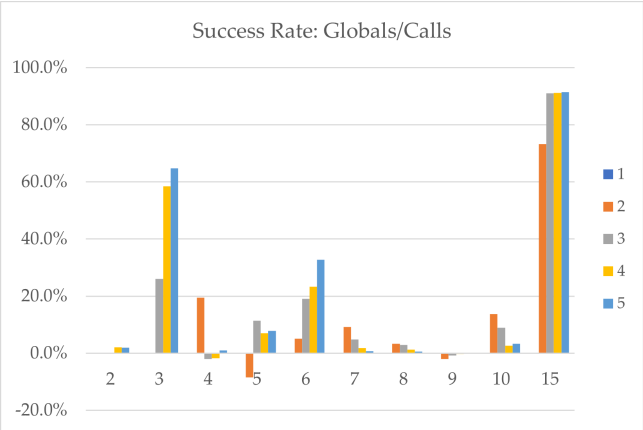


(**e**)

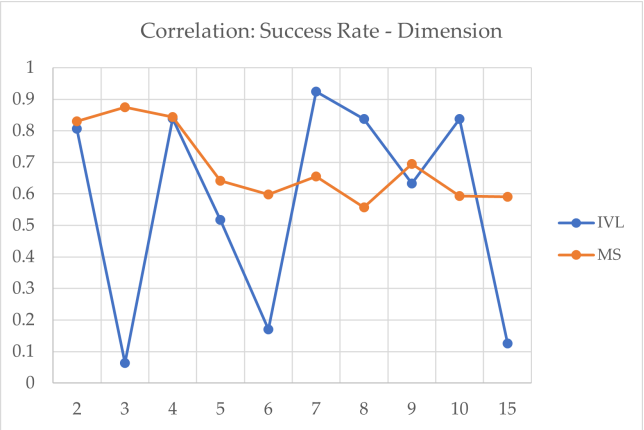


(**f**)

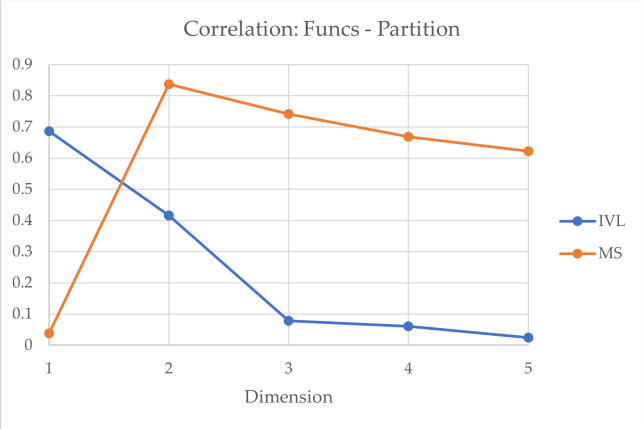
Figure 8. Rastrigin Function: (**a**) Success Rate, (**b**) Correlation: Success Rate−Partition, (**c**) Correlation: Function Evaluations−Partition, (**d**) Correlation: Calls/Locals−Partition, (**e**) Function Evaluations/Dimension, and (**f**) Improvement.



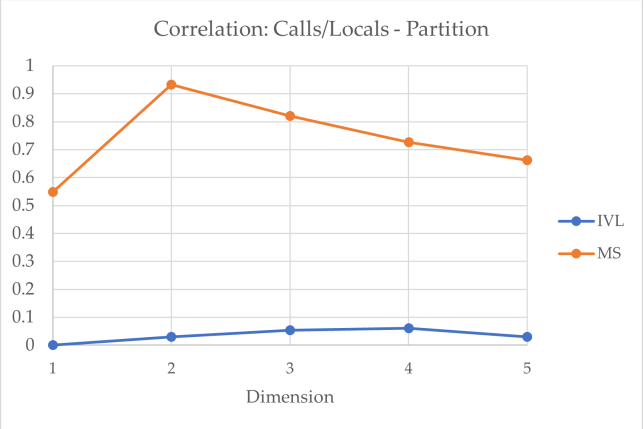
(**a**)



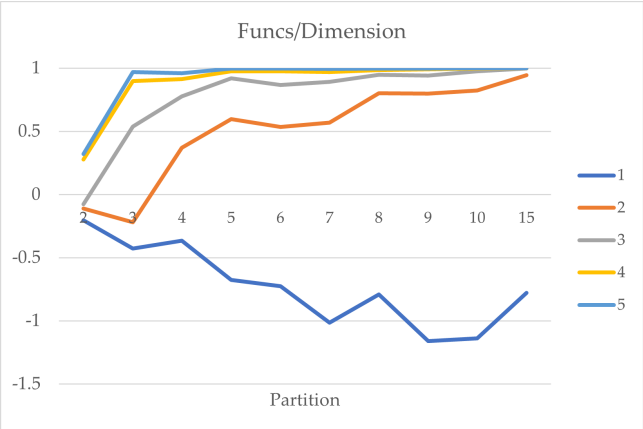
(**b**)



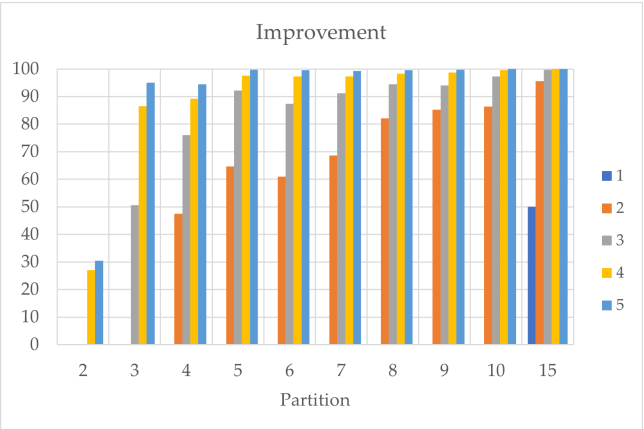
(**c**)



(**d**)



(**e**)



(**f**)

Figure 9. Rosenbrock Function: (**a**) Success Rate, (**b**) Correlation: Success Rate−Partition, (**c**) Correlation: Function Evaluations−Partition, (**d**) Correlation: Calls/Locals−Partition, (**e**) Function Evaluations/Dimension, and (**f**) Improvement.

Table 1. Performance metrics of the Ackley function: success rate, coefficient of variation, and efficiency with respect to dimension and partition size. Entries marked with an asterisk (*) correspond to cases of division by zero.

	2			3			4			5			6			7			8			9			10			15		
	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff
1	0.0%	0.3028	0	0.0%	0.2569	0	*	−0.1525	*	0.0%	0.1509	0	0.0%	0.3671	0	0.0%	0.1955	0	0.0%	0.3364	0	0.0%	−0.0676	0	0.0%	0.3215	0	*	−0.9974	*
2	0.0%	0.4716	0	0.0%	−0.2140	0	*	−0.9871	*	114.3%	−0.2719	3.5918	84.7%	−0.5469	5.5975	293.7%	−0.0925	14.5000	364.7%	−0.0292	24.0223	603.1%	−0.2949	48.4385	272.0%	0.0554	23.9636	944.7%	0.1587	186.7186
3	*	−1	*	*	−0.9630	*	*	−0.2063	*	287.4%	−0.6798	39.6099	−100.0%	−0.4055	−1	523.1%	−0.8216	217.0871	604.9%	1.1320	128.5289	5603.1%	−0.0313	3251.5635	2831.7%	2.0992	1185.9362	3013.2%	−0.4369	8992.6054
4	*	−1	*	7.7%	−0.2622	0.1590	*	0.7140	*	413.0%	−0.7551	322.2085	*	−0.9812	*	585.3%	−0.7877	1650.5924	4012.6%	0.9691	5957.2102	65,600.0%	−0.0374	431,648	3948.1%	3.2576	9568.8784	32,670.2%	−0.9999	1,659,155.57
5	*	−1	*	30.4%	1.7762	0.7007	*	0.5619	*	61.5%	−0.8351	504.4291	−100.0%	−0.3149	−1	9069.1%	−0.6731	154,131.4182	9733.9%	1.1376	64,709.0008	56,678.8%	0.1408	3,352,789.865	310,244.8%	−0.6970	31,034,481.76	128,739.5%	−0.9999	97,838,136.84

Table 2. Preprocessing times (IVL) for the Ackley function.

#	2	3	4	5	6	7	8	9	10	15
1	0.0010	0.0010	0.0010	0.0010	0.0009	0.0011	0.0013	0.0010	0.0011	0.0019
2	0.0013	0.0013	0.0025	0.0036	0.0047	0.0097	0.0081	0.0103	0.0114	0.0255
3	0.0017	0.0046	0.0105	0.0190	0.0309	0.0479	0.0698	0.0977	0.1335	0.4370
4	0.0036	0.0151	0.0418	0.0987	0.2027	0.3764	0.6386	1.0331	1.5492	7.8323
5	0.0083	0.0476	0.1914	0.5750	1.4284	3.0881	6.0089	10.8971	18.2944	139.6668

Table 3. Processing times (left: IVL, right: MS) for the Ackley function.

	2		3		4		5		6		7		8		9		10		15	
	IVL	MS	IVL	MS	IVL	MS	IVL	MS	IVL	MS	IVL	MS	IVL	MS	IVL	MS	IVL	MS	IVL	MS
1	0.0028	0.0023	0.0014	0.0012	0.0014	0.0014	0.0015	0.0014	0.0017	0.0032	0.0015	0.0013	0.0016	0.0012	0.0016	0.0007	0.0021	0.0016	0.0042	0.0021
2	0.0022	0.0028	0.0013	0.0013	0.0022	0.0020	0.0015	0.0025	0.0045	0.0032	0.0033	0.0078	0.0013	0.0055	0.0014	0.0064	0.0022	0.0076	0.0012	0.0174
3	0.0013	0.0010	0.0029	0.0018	0.0025	0.0054	0.0015	0.0102	0.0015	0.0202	0.0015	0.0233	0.0017	0.0414	0.0013	0.0514	0.0075	0.0755	0.0015	0.2476
4	0.0022	0.0014	0.0066	0.0047	0.0183	0.0190	0.0014	0.0403	0.0019	0.0997	0.0013	0.1524	0.0111	0.3088	0.0014	0.4829	0.0018	0.7626	0.0018	3.8704
5	0.0037	0.0023	0.0155	0.0131	0.0012	0.0788	0.0016	0.1937	0.0041	0.5943	0.0016	1.0863	0.0022	2.5584	0.0017	4.4787	0.0020	7.9643	0.0020	60.5127

Table 4. Performance metrics of the Dixon–Price function: success rate, coefficient of variation, and efficiency with respect to dimension and partition size.

	2			3			4			5			6			7			8			9			10			15		
	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff
1	0.0%	0.2101	0	0.0%	0.4199	0	0.0%	0.4399	0	0.0%	0.2797	0	0.0%	1.3908	0	0.0%	0.5682	0	0.0%	0.3316	0	0.0%	0.7618	0	0.0%	0.3039	0	0.0%	−0.6575	1
2	0.0%	0.2229	0	0.0%	0.4751	0	0.0%	−0.5037	1	−27.0%	0.8858	1.1892	0.0%	−0.9343	3	−51.4%	0.5272	1.4289	4.8%	−0.9999	6.3353	−23.2%	0.6331	5.9159	3.8%	−1	9.3842	−22.0%	0.6538	16.2592
3	0.0%	0.6118	0	−19.5%	0.0600	1.4161	16.9%	−0.9999	7.1803	−59.9%	0.0388	4.2193	14.2%	−0.9999	24.1298	−36.6%	0.1806	19.1703	18.5%	−0.9999	60.6085	−34.7%	0.2013	44.8684	−10.3%	0.1569	86.0603	−20.8%	0.2352	244.6160
4	−31.6%	0.4245	0.3684	−100.0%	−0.9999	−1	39.4%	−0.9999	35.2547	−70.2%	−0.2633	17.7438	19.2%	−0.2470	145.2101	−49.0%	−0.0306	103.9915	11.8%	−0.1052	435.6865	−28.3%	0.0238	431.3220	−5.2%	0.0082	901.5030	−8.8%	−0.0777	3470.5174
5	1.3%	−0.0625	1.7022	−100.0%	−0.9998	−1	−12.1%	0.0108	64.1526	−97.1%	−0.7973	7.9559	44.5%	−0.1005	984.8924	−82.2%	−0.4400	243.7062	−10.9%	0.0142	2261.8978	−4.9%	0.0079	4969.5842	−66.5%	−0.1214	1992.7044	10.9%	−0.0598	55,054.0632

Table 5. Preprocessing times (IVL) for the Dixon–Price function.

#	2	3	4	5	6	7	8	9	10	15
1	0.0042	0.0060	0.0017	0.0007	0.0012	0.0013	0.0006	0.0004	0.0003	0.0027
2	0.0023	0.0009	0.0029	0.0020	0.0027	0.0032	0.0045	0.0058	0.0064	0.0145
3	0.0011	0.0031	0.0071	0.0129	0.0213	0.0333	0.0498	0.0698	0.0944	0.3144
4	0.0029	0.0122	0.0334	0.0778	0.1593	0.2972	0.5046	0.7826	1.1926	5.9960
5	0.0069	0.0393	0.1585	0.4783	1.1812	2.5473	4.9776	8.9981	15.2041	115.7074

Table 6. Processing times (left: IVL, right: MS) for the Dixon–Price function.

	2		3		4		5		6		7		8		9		10		15	
	IVL	MS	IVL	MS	IVL	MS	IVL	MS	IVL	MS	IVL	MS	IVL	MS	IVL	MS	IVL	MS	IVL	MS
1	0.4285	0.0204	0.0227	0.0053	0.0283	0.0023	0.0010	0.0008	0.0022	0.0018	0.0009	0.0007	0.0009	0.0005	0.0015	0.0006	0.0008	0.0005	0.0021	0.0015
2	0.1637	0.0083	0.0065	0.0024	0.0046	0.0050	0.0051	0.0036	0.0012	0.0037	0.0016	0.0051	0.0012	0.0067	0.0014	0.0081	0.0012	0.0092	0.0033	0.0198
3	0.0025	0.0021	0.0020	0.0038	0.0013	0.0078	0.0014	0.0147	0.0020	0.0241	0.0015	0.0400	0.0022	0.0567	0.0013	0.0781	0.0014	0.1073	0.0012	0.3617
4	0.0036	0.0028	0.0016	0.0122	0.0018	0.0342	0.0017	0.0821	0.0017	0.1793	0.0016	0.3245	0.0012	0.5490	0.0016	0.8422	0.0016	1.3149	0.0012	6.5467
5	0.0078	0.0063	0.0009	0.0336	0.0015	0.1590	0.0015	0.4743	0.0020	1.1872	0.0010	2.5305	0.0021	5.1153	0.0017	9.0007	0.0017	15.5621	0.0049	116.4989

Table 7. Performance metrics of the Levy function: success rate, coefficient of variation, and efficiency with respect to dimension and partition Size. Entries marked with an asterisk (*) correspond to cases of division by zero.

	2			3			4			5			6			7			8			9			10			15		
	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff
1	*	−0.0676	*	0.0%	0.1324	0	0.0%	0.1556	0	0.0%	0.2399	0	0.0%	0.3953	0	0.0%	0.2721	0	0.0%	0.3701	0	0.0%	0.1437	0	0.0%	0.2959	0	*	−0.9276	*
2	*	−0.4865	*	0.0%	0.0043	0	*	−0.6722	*	*	−0.6993	*	105.1%	−0.6165	3.208	244.8%	−0.6821	10.891	376.2%	−0.7314	21.676	433.9%	−0.8396	39.721	456.0%	−0.7813	32.295	418.9%	−0.9343	102.784
3	*	−0.4845	*	*	−0.7589	*	276.3%	−0.7301	13.163	278.4%	−0.7719	21.667	1111.1%	−0.8647	185.328	*	−0.8141	*	1834.5%	−0.9076	597.781	1467.9%	−0.9153	799.413	3728.2%	−0.9046	2501.090	833.0%	−0.9594	2482.020
4	*	−0.7280	*	*	−0.6122	*	753.7%	−0.8434	142.199	−100.0%	−0.8335	−1	6552.0%	−0.9506	6501.001	−100.0%	−0.8272	−1	6645.5%	−0.9522	15,624.203	4194.6%	−0.9718	15,850.317	26,725.6%	−0.9506	162,578.596	2876.3%	−0.9726	78,482.737
5	*	−0.7962	*	*	−0.6179	*	2289.3%	−0.9247	1863.378	*	−0.8673	*	28,502.9%	−0.9782	186,999.741	−100.0%	−0.8902	−1	11,134.7%	−0.9745	212,808.556	3411.3%	−0.9822	77,947.323	229,160.9%	−0.9664	11,756,970.03	8257.6%	−0.9865	2,677,901.014

Table 8. Preprocessing times (IVL) for the Levy function.

#	2	3	4	5	6	7	8	9	10	15
1	0.0081	0.0033	0.0243	0.0026	0.0025	0.0046	0.0038	0.0022	0.0039	0.0052
2	0.0024	0.0024	0.0082	0.0079	0.0111	0.0168	0.0178	0.0218	0.0228	0.0438
3	0.0026	0.0082	0.0184	0.0321	0.0523	0.0826	0.1231	0.1690	0.2375	0.7531
4	0.0063	0.0260	0.0766	0.1930	0.3756	0.6799	1.1643	1.8409	2.8630	14.1993
5	0.0143	0.0839	0.3540	1.1133	2.6624	5.7357	11.3356	20.1256	34.8130	259.5128

Table 9. Processing times (left: IVL, right: MS) for the Levy function.

	2		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**		**15**	
	IVL	**MS**	**IVL**	**MS**	**IVL**	**MS**	**IVL**	**MS**	**IVL**	**MS**	**IVL**	**MS**	**IVL**	**MS**	**IVL**	**MS**	**IVL**	**MS**	**IVL**	**MS**
1	0.0383	0.0016	0.0012	0.0011	0.0032	0.0012	0.0011	0.0010	0.0013	0.0019	0.0014	0.0010	0.0017	0.0009	0.0013	0.0009	0.0032	0.0008	0.0030	0.0019
2	0.0244	0.0016	0.0039	0.0012	0.0065	0.0030	0.0057	0.0029	0.0055	0.0034	0.0013	0.0043	0.0026	0.0063	0.0011	0.0070	0.0016	0.0076	0.0014	0.0182
3	0.0016	0.0009	0.0042	0.0030	0.0016	0.0070	0.0060	0.0121	0.0031	0.0187	0.0019	0.0306	0.0013	0.0608	0.0024	0.0624	0.0059	0.0775	0.0012	0.3176
4	0.0026	0.0017	0.0089	0.0086	0.0015	0.0299	0.0048	0.0630	0.0016	0.1203	0.0055	0.2521	0.0040	0.4156	0.0025	0.6420	0.0046	0.8696	0.0016	5.4197
5	0.0043	0.0032	0.0223	0.0281	0.0030	0.1296	0.0248	0.3637	0.0018	0.8279	0.0342	1.9264	0.0017	3.8885	0.0028	6.8542	0.0031	10.0263	0.0015	98.1058

Table 10. Performance metrics of the Rastrigin function: success rate, coefficient of variation, and efficiency with respect to dimension and partition size. Entries marked with an asterisk (*) correspond to cases of division by zero.

	2			**3**			**4**			**5**			**6**			**7**			**8**			**9**			**10**			**15**		
	SR	**CV**	**Eff**	**SR**	**CV**	**Eff**	**SR**	**CV**	**Eff**	**SR**	**CV**	**Eff**	**SR**	**CV**	**Eff**	**SR**	**CV**	**Eff**	**SR**	**CV**	**Eff**	**SR**	**CV**	**Eff**	**SR**	**CV**	**Eff**	**SR**	**CV**	**Eff**
1	*	−1	*	0.0%	−0.0363	0	0.0%	0.2486	0	0.0%	−0.0497	0	0.0%	0.3011	0	0.0%	0.4904	0	0.0%	0.3343	0	0.0%	0.4644	0	*	0.1872	*	*	−0.2102	*
2	*	−1	*	0.0%	−0.0392	0	*	−0.6327	*	*	−0.3818	*	−100.0%	−0.4764	−1	33.9%	−0.4360	3.6482	−100.0%	−0.8251	−1	631.0%	−0.9313	62.8699	*	−0.7318	*	477.6%	−0.9397	95.2596
3	*	−1	*	*	−0.6019	*	59.6%	−0.3894	1.7248	1200.0%	−0.9406	168	−100.0%	−0.1099	−1	463.5%	−0.8976	178.2974	−100.0%	−0.8786	−1	2608.8%	−0.9840	1937.6315	*	−0.7306	*	995.5%	−0.9509	1002.4984
4	*	−1	*	286.3%	−0.5862	13.9201	410.9%	−0.6144	43.5718	5715.4%	−0.9499	3662.6923	−100.0%	−0.3598	−1	980.2%	−1	2602.3617	−100.0%	−0.8905	−1	8133.1%	−1	54,090.3534	*	−0.8179	*	1404.5%	−0.9601	7685.3858
5	*	−1	*	996.5%	−0.4524	119.2293	1185.2%	−0.8228	555.2022	31200.0%	−0.9645	97,968	−100.0%	−0.5276	−1	1642.0%	−1	29,281.4974	−100.0%	−0.9605	−1	24,170.4%	−1	1,433,168.955	*	−0.8837	*	4738.3%	−0.9634	165,202.7248

Table 11. Preprocessing times (IVL) for the Rastrigin function.

#	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**15**
1	0.0018	0.0012	0.0006	0.0006	0.0008	0.0006	0.0006	0.0007	0.0007	0.0015
2	0.0010	0.0009	0.0020	0.0026	0.0041	0.0043	0.0059	0.0075	0.0083	0.0193
3	0.0012	0.0037	0.0084	0.0154	0.0259	0.0412	0.0586	0.0812	0.1063	0.3752
4	0.0031	0.0137	0.0362	0.0862	0.1784	0.3328	0.5546	0.8836	1.3339	7.2134
5	0.0075	0.0429	0.1706	0.5211	1.2986	2.8194	5.4560	9.8211	16.4495	133.8370

Table 12. Processing times (left: IVL, right: MS) for the Rastrigin function.

	2		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**		**15**	
	IVL	**MS**	**IVL**	**MS**	**IVL**	**MS**	**IVL**	**MS**	**IVL**	**MS**	**IVL**	**MS**	**IVL**	**MS**	**IVL**	**MS**	**IVL**	**MS**	**IVL**	**MS**
1	0.0156	0.0017	0.0028	0.0012	0.0012	0.0011	0.0040	0.0017	0.0014	0.0019	0.0011	0.0011	0.0014	0.0010	0.0011	0.0009	0.0012	0.0009	0.0020	0.0020
2	0.0146	0.0016	0.0029	0.0015	0.0037	0.0018	0.0016	0.0025	0.0031	0.0035	0.0013	0.0041	0.0013	0.0054	0.0045	0.0074	0.0013	0.0076	0.0042	0.0149
3	0.0013	0.0010	0.0014	0.0029	0.0060	0.0045	0.0015	0.0085	0.0033	0.0161	0.0015	0.0277	0.0023	0.0350	0.0012	0.0535	0.0238	0.0730	0.0013	0.2074
4	0.0022	0.0014	0.0042	0.0072	0.0011	0.0154	0.0012	0.0369	0.0096	0.0926	0.0014	0.1818	0.0163	0.2816	0.0011	0.4642	0.0187	0.7193	0.0072	3.0977
5	0.0043	0.0029	0.0013	0.0192	0.0010	0.0597	0.0012	0.1880	0.0182	0.5618	0.0014	1.2696	0.1069	2.2625	0.0012	4.2861	0.1176	7.4535	0.0060	47.5542

Table 13. Performance metrics of the Rosenbrock function: success rate, coefficient of variation, and efficiency with respect to dimension and partition Size. Entries marked with an asterisk (*) correspond to cases of division by zero.

	2			3			4			5			6			7			8			9			10			15		
	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff	SR	CV	Eff
1	0.0%	0.3543	0	0.0%	*	0	0.0%	0.4674	0	0.0%	*	0	0.0%	0.5437	0	0.0%	0.4021	0	0.0%	0.2832	0	0.0%	0.8290	0	0.0%	0.5701	0	0.0%	−0.4031	1
2	0.0%	0.3864	0	0.0%	0.4482	0	90.5%	0.0509	2.6281	−39.0%	−0.5509	0.7252	156.4%	−0.8599	5.5746	218.5%	−0.8308	9.1424	460.0%	−0.8535	30.36	−31.2%	−0.9366	3.6567	291.1%	−0.8911	27.7584	372.2%	−0.9743	107.6026
3	0.0%	0.3400	0	102.7%	−0.0446	3.1088	−25.6%	−0.7033	2.1002	100.8%	−0.9075	24.5895	517.7%	−0.8187	48.0616	734.5%	−0.9104	93.2180	1711.8%	−0.9435	327.2788	−63.9%	−0.9500	5.0564	1810.9%	−0.9357	714.6842	1007.5%	−1	3742.2503
4	37.0%	−0.2930	0.8765	279.8%	−0.6191	27.2506	−36.0%	−0.8716	4.9664	94.4%	−0.8903	82.3076	1680.3%	−0.8913	647.2767	1834.4%	−0.9469	716.2339	5634.3%	−0.9496	3287.1804	−100.0%	−0.9715	−1	4023.7%	−0.9702	8501.4976	1031.2%	−1	57,272.6533
5	43.9%	−0.5129	1.0703	476.5%	−0.7817	114.2941	409.5%	−0.8928	91.7256	138.0%	−0.9698	715.4350	8154.0%	−0.9261	22,374.0705	7806.9%	−0.9604	12,502.7103	22,376.0%	−0.9473	50,516.0329	−100.0%	−0.9938	−1	85,340.9%	−0.9842	1,849,368.5870	1059.9%	−1	880,807.0508

Table 14. Preprocessing times (IVL) for the Rosenbrock function.

#	2	3	4	5	6	7	8	9	10	15
1	0.0013	0.0006	0.0006	0.0006	0.0007	0.0006	0.0007	0.0007	0.0006	0.0016
2	0.0012	0.0009	0.0021	0.0027	0.0034	0.0043	0.0063	0.0077	0.0094	0.0186
3	0.0012	0.0038	0.0087	0.0161	0.0256	0.0418	0.0622	0.0772	0.1061	0.3539
4	0.0031	0.0135	0.0368	0.0870	0.1785	0.3295	0.6086	0.8891	1.3613	6.8458
5	0.0076	0.0427	0.1740	0.5273	1.3007	2.7937	6.0006	9.8732	16.8640	127.6426

Table 15. Processing times (left: IVL, right: MS) for the Rosenbrock function.

	2		3		4		5		6		7		8		9		10		15	
	IVL	MS	IVL	MS	IVL	MS	IVL	MS	IVL	MS	IVL	MS	IVL	MS	IVL	MS	IVL	MS	IVL	MS
1	0.0184	0.0019	0.0012	0.0011	0.0012	0.0013	0.0010	0.0012	0.0015	0.0018	0.0011	0.0009	0.0011	0.0007	0.0011	0.0007	0.0011	0.0007	0.0015	0.0020
2	0.0163	0.0017	0.0015	0.0014	0.0024	0.0023	0.0017	0.0032	0.0010	0.0028	0.0014	0.0037	0.0012	0.0057	0.0015	0.0069	0.0013	0.0076	0.0012	0.0184
3	0.0011	0.0008	0.0057	0.0028	0.0012	0.0060	0.0012	0.0113	0.0012	0.0147	0.0050	0.0255	0.0010	0.0383	0.0042	0.0524	0.0050	0.0618	0.0012	0.2742
4	0.0016	0.0019	0.0008	0.0064	0.0030	0.0191	0.0011	0.0523	0.0010	0.0779	0.0069	0.1662	0.0090	0.3136	0.0074	0.4934	0.0021	0.6169	0.0022	4.1540
5	0.0032	0.0029	0.0009	0.0184	0.0055	0.0809	0.0010	0.2797	0.0011	0.4615	0.0062	1.1921	0.0178	2.6545	0.0146	4.6996	0.0055	6.2630	0.0012	67.2266

3.1. Limitations

3.1.1. On Methodological Comparisons

In evaluating the performance of the proposed framework, we chose to compare it with the conventional multistart approach, as both share a common methodological foundation: the use of multiple starting points to guide local optimization. This enables a fair and consistent basis for comparison in terms of sampling strategies, coverage, and convergence behavior. While numerous other global optimization techniques exist—such as evolutionary algorithms, swarm intelligence methods, and Bayesian Optimization—a direct comparison with these would not be methodologically appropriate, as they operate under fundamentally different principles.

For instance, BO employs surrogate probabilistic models—such as Gaussian Processes—to approximate the objective function, along with acquisition functions to identify promising regions [49]. In contrast, our framework constructs boxes around sampled points and uses interval arithmetic to deterministically enclose the objective function's ranges withing these boxes. These enclosures are then used to assess the boxes, determine the most promising one, and facilitate pruning through rigorous discard criteria, such as the cut-off test. As such, rather than presenting cross-paradigm benchmarking, we focused on demonstrating the strengths and theoretical properties of our framework within its intended design space.

3.1.2. On High-Dimensional Spaces

The curse of dimensionality [50] is a well-known challenge in global optimization, particularly for sampling-based methods. In the proposed framework, the percentage coverage of the search space is exponentially inversely proportional to the dimensionality d (Corollary 1). To address this, we introduce a compensatory mechanism: expanding the interval boxes to ensure that a predefined coverage rate is maintained even as d increases.

However, while this strategy ensures theoretical coverage of the search space, it incurs an increasing computational cost. Specifically, the required sample size $S \sim N^d$ increases exponentially with dimension. Consequently, the practical applicability of the framework is naturally restricted beyond a certain dimensional threshold—typically $d \sim 10$–20 with $N \sim 2$–5, under common computational budgets.

Table 16 illustrates how sample size scales with dimensionality and partitioning, alongside the corresponding number of local optimizer calls, for the Rastrigin, Dixon–Price, and Rosenbrock benchmark functions. The results highlight both the efficiency of the framework and the computational limits imposed by the number of interval evaluations required during the preprocessing stage. Beyond this threshold, the volume of interval calculations becomes prohibitively expensive.

To extend the framework's applicability in higher dimensions, alternative strategies must be explored, such as dimensionality reduction, problem decomposition, or hierarchical sampling. Importantly, the limitation lies not in the prioritization mechanism itself but in the cost of obtaining reliable interval bounds for each partition. Once these bounds are available, the framework exhibits extremely efficient behavior, especially in high dimensions: the selection of regions based on their lower bounds guides the optimizer toward the most promising areas with minimal waste. Therefore, future improvements in approximating or accelerating this preprocessing step—without compromising the integrity of the bounds—could significantly enhance the scalability and practicality of the method in high-dimensional settings.

Table 16. Sample size (S) and framework's function calls performed (for a 10% space coverage) with respect to dimension (d) and partitioning (N = 2, 3, and 4, for the Rastrigin, Dixon–Price, and Rosenbrock benchmark functions, respectively).

N	2		3		4	
d	S	Calls	S	Calls	S	Calls
1	1	1	1	1	1	1
2	1	1	1	1	2	1
3	1	1	3	3	7	1
4	2	1	9	1	26	3
5	4	2	25	4	103	2
6	7	1	73	11	410	7
7	13	1	219	2	1639	52
8	26	13	657	3	6554	179
9	52	26	1969	1	26,215	147
10	103	51	5905	20	104,858	416
11	205	102	17,715	1	419,431	—
12	410	205	53,145	3	1,677,722	—
13	820	410	159,433	3	6,710,887	—
14	1639	820	478,297	2	26,843,546	—
15	3277	1539	1,434,891	12	107,374,183	—
16	6554	3277	4,304,673	—	429,496,730	—
17	13,108	6554	12,914,017	—	1.7×10^9	—
18	26,215	13,107	38,742,049	—	6.9×10^9	—
19	52,429	26,214	116,226,147	—	2.7×10^{10}	—
20	104,858	52,429	348,678,441	—	1.1×10^{11}	—

4. Discussion

The core idea behind the proposed framework is to move beyond real arithmetic by treating regions of the search space as entities to be analyzed and prioritized. Instead of evaluating the objective function at individual points, the method computes interval enclosures—mathematically guaranteed bounds—over subregions. Each such region is associated with a lower and upper bound, enclosing the possible function values within it. This enables two key capabilities: (a) discarding regions that do not contain the global minimum with certainty, and (b) assessing the remaining regions based on their lower bounds to guide the local optimizer toward the most promising areas. This rigorous approach not only enhances the efficiency of the search but also provides a mechanism for verifying whether a candidate solution is globally optimal within a specified tolerance.

Numerical experiments performed on standard benchmark functions confirm the method's effectiveness, especially in higher-dimensional settings. The combination of Latin Hypercube Sampling and Interval Arithmetic enables a more informed and structured sampling of the search space, resulting in improvements across all considered performance metrics.

The presented numerical results demonstrate an increased success rate for the proposed method, particularly as the dimensionality of the objective function increases. Figures 5–9, Tables 1–15 and A1–A5 (in Appendix A) show that, in most cases, the proposed multistart framework significantly outperforms the conventional multistart method in terms of both efficiency and effectiveness. Moreover, the proposed framework achieves closer proximity to the global minimum, as reflected in the reduced standard deviations of proximity and the number of function evaluations.

The utilization of acceleration devices discards regions that do not contain the global minimum with certainty at the beginning of the optimization process. Additionally, the satisfaction of the strong termination criterion suggests that the algorithm is capable of identi-

fying the global minimum with arbitrary precision, an important advantage in problems where verification of solutions is costly or impractical.

5. Conclusions

This work presented a selective multistart global optimization framework based on interval arithmetic and stratified sampling. Each sampling point is enclosed in an interval box and assessed based on the lower bound of the function's interval enclosure. The local optimization process is initiated from the most promising sample points, while both strong and weak termination criteria are applied to terminate the search process. Numerical experiments on well-known test functions showed that the method is effective in finding both global and local minima using fewer function evaluations. This is supported by the reported distance to the global minimum, success rates, and the number of global minima found per optimizer call.

The main contribution of the proposed framework lies in its ability to quantify and manage space coverage—a feature that is usually assumed but not controlled in standard multistart methods. Additionally, the use of interval arithmetic enables the method to discard with certainty regions that cannot contain the global minimum, leveraging techniques from interval optimization to accelerate the search.

However, the current framework is limited to unconstrained, continuous optimization problems. Extending the method to constrained or combinatorial problems would require significant modifications, such as constraints handling and adaptation of interval-based methodologies. While the interval preprocessing cost remains low, a more detailed analysis of runtime and scalability in high-dimensional settings is warranted and left for future work.

The coverage issue associated with LHS in high-dimensional spaces remains a fundamental challenge, as the coverage rate decreases exponentially with increasing dimensionality. Although the proposed framework mitigates this limitation to some extent by adjusting the sample size and the corresponding interval enclosures, ensuring sufficient exploration of the search space remains a key issue that requires further investigation.

Future research should also include combining this framework with other global optimization strategies (e.g., Differential Evolution, Bayesian Optimization), potentially implementing adaptive box sizing and refined sampling techniques, as well as applying this method to real-world problems—such as hyperparameter tuning or engineering design, which often exhibit additional complexities such as noise, discontinuities, or high computational costs. Finally, addressing the coverage limitations in high dimensions, and scaling the algorithm—potentially through parallelization, advanced prioritization, and tighter bounding techniques—will also be critical for advancing the method's practical applicability and robustness.

Author Contributions: Conceptualization, I.A.N. and V.P.G.; methodology, I.A.N. and V.P.G.; software, V.P.G. and I.A.N.; validation, V.P.G., I.A.N. and V.C.L.; formal analysis, I.A.N. and V.P.G.; investigation, V.P.G. and I.A.N.; data curation, V.P.G.; writing—original draft preparation, V.P.G. and I.A.N.; writing—review and editing, V.P.G., I.A.N. and V.C.L.; visualization, I.A.N. and V.P.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

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

Appendix A

In the following tables, seven (7) ratios are presented. The columns represent the partition size (2, 3, 4, 5, 6, 7, 8, 9, 10, 15), grouped together for each dimension (1, 2, 3, 4, 5). Our proposed framework is referred as "IVL" while the conventional one as "MS". All ratios are calculated in such a way where a value >1 indicates that our framework yields better results than the conventional one, while <1 means that it yields worse.

The calculated ratios, in the presented order, are the following:

1. [Globals Detected/Calls (IVL)]/[Globals Detected/Calls (MS)]
2. [Locals Detected/Calls (IVL)]/[Locals Detected/Calls (MS)]
3. Average Distance (MS)/Average Distance (IVL)
4. St. Dev. of Distance (MS)/St. Dev. of Distance (IVL)
5. Calls (MS)/Calls (IVL)
6. Function Evaluations (MS)/Function Evaluations (IVL)
7. Strong Criterion/Calls (IVL)

The first ratio represents efficiency regarding globals detection. The second ratio represents efficiency regarding locals detection. The third ratio represents the proximity of each method to the global minimum, since each "distance" is the distance of each detected minimum to the known global minimum. The forth ratio represents reliability regarding global minimum proximity. The fifth ratio represents efficiency regarding function calls (a measure of computation cost). The sixth ratio represents efficiency regarding function computations (another measure of computation cost). Lastly, the seventh ratio is not a comparison between methods; it represents the frequency of strongly detected minimums (1st criterion) of our proposed framework.

Table A1. Ratio metrics of the Ackley function.

#	2	3	4	5	6	7	8	9	10	15
1	1	1	-	1	1	1	1	1	1	-
	1	1	1	1	1	1	1	1	1	1
	0.35	0.32	0.35	0.34	0.35	0.39	0.38	0.35	0.32	0.53
	0.77	0.8	1.2	0.87	0.73	0.84	0.75	1.1	0.76	3.8×10^2
	1	1	1	1	1	1	1	1	1	1.1
	0.94	0.81	0.74	0.7	0.76	0.73	0.76	0.55	0.69	0.62
	1	0.4	0	0.13	0.38	0.43	0.56	0.17	0.45	0
2	1	1	-	2.1	1.8	3.9	4.6	7	3.7	10
	1	1	1	1	1	1	1	1	1	1
	0.35	0.35	0.54	1.5	1.7	1.5	2.3	1.8	1.8	4.5
	0.68	1.3	77	1.4	2.2	1.1	1	1.4	0.95	0.86
	1	1	1	2.1	3.6	3.9	5.4	7	6.7	18
	0.97	0.82	0.87	1.5	2.7	2.3	3.5	4.4	4.8	8.9
	1	0.15	0	0.0071	0.13	0.27	0.48	0.1	0.31	0.59
3	-	-	-	3.9	0	6.2	7	57	29	31
	1	1	1	1	1	0.94	1	1	1	1
	0.35	0.63	1.2	23	1.4	17	3	2.7	3.5	44
	1.3×10^{15}	27	1.3	3.1	1.7	5.6	0.47	1	0.32	1.8
	1	1	1.9	10	18	35	18	57	40	2.9×10^2
	0.9	0.93	1.8	4.8	26	15	19	42	31	1.3×10^2
	0	0	0	0.74	0	0.53	0.22	0.039	0.26	0.79
4	-	1.1	-	5.1	-	6.9	41	6.6×10^2	40	3.3×10^2
	1	1	1	1	1	1	1	1	1	1
	0.54	0.99	2.1	8.9	1.5	9.7	10	3.5	5.7	7.1×10^6
	1.3×10^{15}	1.4	0.58	4.1	53	4.7	0.51	1	0.23	7.2×10^5
	1	1.1	5.1	63	76	2.4×10^2	1.4×10^2	6.6×10^2	2.4×10^2	5.1×10^3
	0.96	1.1	4.7	26	1.6×10^2	1.4×10^2	1.3×10^2	5.1×10^2	2.6×10^2	2.2×10^3
	0	0.0096	0	0.5	0	0.12	0.31	0.1	0.18	1

Table A1. *Cont.*

#	2	3	4	5	6	7	8	9	10	15
	-	1.3	-	1.6	0	92	98	5.7×10^2	3.1×10^3	1.3×10^3
	1	1	1	1	1	1	1	1	1	1
	0.69	1.2	2.8	6.7	2.2	11	22	4.1	6×10^2	2.8×10^6
5	8.3×10^{14}	0.36	0.64	6.1	1.5	3.1	0.47	0.88	3.3	7.2×10^4
	1	1.3	23	3.1×10^2	2.1×10^2	1.7×10^3	6.6×10^2	5.9×10^3	1×10^4	7.6×10^4
	0.98	1.3	26	1.3×10^2	2.6×10^2	1.1×10^3	6.4×10^2	4.6×10^3	3.9×10^3	4.2×10^4
	0	0.0068	0	0.06	0	0.06	0.19	0.1	0.99	1

Table A2. Ratio metrics of the Dixon–Price function.

#	2	3	4	5	6	7	8	9	10	15
	1	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1	1
	0.33	0.32	0.38	0.37	0.18	0.37	0.37	0.34	0.41	0.27
1	0.83	0.7	0.69	0.78	0.42	0.64	0.75	0.57	0.77	2.9
	1	1	1	1	1	1	1	1	1	2
	0.6	0.47	0.43	0.35	0.33	0.29	0.27	0.24	0.22	0.38
	1	1	1	1	1	1	1	1	1	1
	1	1	1	0.73	1	0.49	1	0.77	1	0.78
	1	1	1	1	1	1	1	1	1	1
	0.36	0.38	3.9	0.19	29	0.2	1.6×10^6	0.28	3.3×10^8	0.3
2	0.82	0.68	2	0.53	15	0.65	2.7×10^6	0.61	2.2×10^9	0.6
	1	1	2	3	4	5	7	9	10	22
	0.95	0.92	1.6	3.4	4.3	7.4	7.9	11	11	25
	1	0.75	1	0.63	1	0.41	1	0.66	1	0.71
	1	0.81	1.2	0.4	1.1	0.63	1.2	0.65	0.9	0.79
	1	1	1	1	1	1	1	1	1	1
	0.3	0.52	1.4×10^6	0.37	2.4×10^7	0.49	1.1×10^9	0.48	0.69	0.55
3	0.62	0.94	8.5×10^5	0.96	8.6×10^6	0.85	1.1×10^9	0.83	0.86	0.81
	1	3	7	13	22	32	52	70	97	3.1×10^2
	0.98	3	7.5	23	24	52	53	1.2×10^2	1.2×10^2	5.2×10^2
	1	0.4	1	0.29	1	0.47	1	0.51	0.76	0.63
	0.68	0	1.4	0.3	1.2	0.51	1.1	0.72	0.95	0.91
	1	1	0.98	1	1	1	1	1	1	1
	0.3	0.49	1.1×10^7	0.59	2.2	0.62	1.5	0.74	0.99	1.4
4	0.7	1.1×10^4	3.8×10^6	1.4	1.3	1	1.1	0.98	0.99	1.1
	2	9	26	63	1.2×10^2	2.1×10^2	3.9×10^2	6×10^2	9.5×10^2	3.8×10^3
	2.2	14	30	1.2×10^2	1.5×10^2	3.5×10^2	5.2×10^2	1.1×10^3	1.3×10^3	7.8×10^3
	0.52	0	1	0.16	0.81	0.29	0.73	0.42	0.62	0.56
	1	0	0.88	0.029	1.4	0.18	0.89	0.95	0.33	1.1
	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
	1.2	0.63	1.1	0.57	1.8	0.64	1.1	0.98	0.67	1.7
5	1.1	9.1×10^3	0.99	4.9	1.1	1.8	0.99	0.99	1.1	1.1
	2.7	25	74	3.1×10^2	6.8×10^2	1.4×10^3	2.5×10^3	5.2×10^3	6×10^3	5×10^4
	2.8	47	97	6.8×10^2	9.4×10^2	2.3×10^3	4.1×10^3	1×10^4	1.1×10^4	1.2×10^5
	0.51	0	0.39	0.01	0.64	0.066	0.4	0.37	0.15	0.45

Table A3. Ratio metrics of the Levy function.

#	2	3	4	5	6	7	8	9	10	15
	-	1	1	1	1	1	1	1	1	-
	1	1	1	1	1	1	1	1	1	1
	0.34	0.33	0.33	0.35	0.33	0.38	0.36	0.35	0.34	0.75
1	1.1	0.88	0.87	0.81	0.72	0.79	0.73	0.87	0.77	14
	1	1	1	1	1	1	1	1	1	2
	0.75	0.66	0.63	0.57	0.54	0.47	0.49	0.4	0.36	0.57
	0	0.32	0.18	0.25	0.37	0.36	0.48	0.16	0.31	0

Table A3. *Cont.*

#	2	3	4	5	6	7	8	9	10	15
	-	1	-	-	2.1	3.4	4.8	5.3	5.6	5.2
	1	1	1	1	1	1	1	1	1	1
	0.35	0.36	0.64	0.97	1.5	2.6	5.9	12	7.6	1×10^2
2	1.9	1	3.1	3.3	2.6	3.1	3.7	6.2	4.6	15
	1	1	1	1.4	2.1	3.4	4.8	7.6	6	20
	0.87	0.84	0.89	1.2	1.7	2.7	3.4	5	4.3	12
	0	0.15	0	0	0.021	0.13	0.37	0.59	0.31	0.83
	-	-	3.8	3.8	12	-	19	16	38	9.3
	1	1	1	1	1	1	1	1	1	1
	0.34	0.85	5.2	7.6	15	6.3	34	40	29	55
3	1.9	4.1	3.7	4.4	7.4	5.4	11	12	10	25
	1	1	3.8	6	15	12	31	51	65	2.7×10^2
	0.92	0.97	3.4	5.4	12	10	25	44	58	2×10^2
	0	0	0.15	0.055	0.26	0	0.27	0.3	0.27	0.3
	-	-	8.5	0	67	0	67	43	2.7×10^2	30
	1	1	1	1	1	1	1	1	1	1
	0.65	2	17	12	60	7.5	92	89	46	94
4	3.7	2.6	6.4	6	20	5.8	21	35	20	37
	1	1.1	17	16	98	41	2.3×10^2	3.7×10^2	6.1×10^2	2.6×10^3
	0.97	1.1	18	15	79	38	2.2×10^2	4.5×10^2	6.3×10^2	2.7×10^3
	0	0	0.37	0	0.37	0	0.3	0.21	0.16	0.23
	-	-	24	-	2.9×10^2	0	1.1×10^2	35	2.3×10^3	84
	1	1	1	1	1	1	1	1	1	1
	0.92	3.3	51	14	1.1×10^2	11	1.2×10^2	94	97	1.7×10^2
5	4.9	2.6	13	7.5	46	9.1	39	56	30	74
	1	1.4	78	39	6.5×10^2	1.6×10^2	1.9×10^3	2.2×10^3	5.1×10^3	3.2×10^4
	0.99	1.4	1.1×10^2	38	6.2×10^2	1.6×10^2	2.2×10^3	3.6×10^3	6.6×10^3	4.4×10^4
	0	0	0.6	0	0.29	0	0.25	0.068	0.19	0.19

Table A4. Ratio metrics of the Rastrigin function.

#	2	3	4	5	6	7	8	9	10	15
	-	1	1	1	1	1	1	1	-	-
	1	1	1	1	1	1	1	1	1	1
	0.35	0.36	0.35	0.35	0.36	0.33	0.36	0.36	0.38	0.6
1	8×10^{14}	1	0.8	1.1	0.77	0.67	0.75	0.68	0.84	1.3
	1	1	1	1	1	1	1	1	1	1.1
	0.83	0.79	0.66	0.66	0.61	0.58	0.61	0.53	0.49	0.52
	0	0.24	0.43	0.23	0.3	0.65	0.31	0.35	0	0
	-	1	-	-	0	1.3	0	7.3	-	5.8
	1	1	1	1	1	1	1	1	1	1
	0.35	0.33	0.57	2.2	0.77	1.7	2.9	49	3.5	18
2	2.4×10^{15}	1	2.7	1.6	1.9	1.8	5.7	15	3.7	17
	1	1	1	1.8	1.4	3.5	4.9	8.7	3.4	17
	0.88	0.87	0.82	1.4	1.4	2.8	4.3	7.1	2.8	9.4
	0	0.12	0	0	0	0.44	0	0.84	0	0.25
	-	-	1.6	13	0	5.6	0	27	-	11
	1	1	1	1	1	1	1	1	1	1
	0.35	0.7	2	12	2.4	1.2×10^2	4.7	3.7×10^2	6.1	28
3	1.8×10^{15}	2.5	1.6	17	1.1	9.8	8.2	62	3.7	20
	1	1.2	1.7	13	3	32	11	72	7	92
	0.91	1.1	1.6	8.4	3.1	23	14	68	7.3	77
	0	0	0.1	0.15	0	0.9	0	0.93	0	0.098
	-	3.9	5.1	58	0	11	0	82	-	15
	1	1	1	1	1	1	1	1	1	1
	0.54	1.7	16	20	4	5.1×10^{12}	6.6	6.6×10^{14}	9.6	35
4	8.3×10^{14}	2.4	2.6	20	1.6	5.8×10^{11}	9.1	1.2×10^{15}	5.5	25
	1	3.9	8.7	63	9	2.4×10^2	24	6.6×10^2	29	5.1×10^2
	0.96	3.7	7.9	44	11	1.8×10^2	31	6.7×10^2	31	4.9×10^2
	0	0.0043	0.3	0.12	0	1	0	1	0	0.024

Table A4. *Cont.*

#	2	3	4	5	6	7	8	9	10	15
	-	11	13	3.1×10^2	0	17	0	2.4×10^2	-	48
	1	1	1	1	1	1	1	1	1	1
	0.69	3	1.1×10^2	26	5.5	5×10^{13}	6.6	5.6×10^{14}	11	53
5	-	1.8	5.6	28	2.1	6.8×10^{12}	25	1.2×10^{15}	8.6	27
	1	11	43	3.1×10^2	30	1.7×10^3	46	5.9×10^3	92	3.4×10^3
	0.99	11	39	2.2×10^2	38	1.2×10^3	60	6.5×10^3	1×10^2	3.4×10^3
	0	0.044	0.41	0.07	0	1	0	1	0	0.016

Table A5. Ratio metrics of the Rosenbrock function.

#	2	3	4	5	6	7	8	9	10	15
	1	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1	1
	0.39	-	0.34	-	0.31	0.32	0.36	0.28	0.32	0.26
1	0.74	-	0.68	-	0.65	0.71	0.78	0.55	0.64	1.7
	1	1	1	1	1	1	1	1	1	2
	0.83	0.7	0.73	0.6	0.58	0.5	0.56	0.46	0.47	0.56
	0.59	1	0.48	1	0.29	0.21	0.1	0.26	0.39	1
	1	1	1.9	0.61	2.6	3.2	5.6	0.69	3.9	4.7
	1	1	1	1	1	1	1	1	1	1
	0.37	0.36	0.66	1.4	2.6	4.3	3.2	5.3	8.4	1.3×10^2
2	0.72	0.69	0.95	2.2	7.1	5.9	6.8	16	9.2	39
	1	1	1.9	2.8	2.6	3.2	5.6	6.8	7.4	23
	0.9	0.82	1.6	2.5	2.1	2.3	5	4.9	5.7	18
	0.28	0.65	0.41	0.13	0.083	0.13	0.04	0.045	0.18	0.93
	1	2	0.74	2	6.2	8.3	18	0.36	19	11
	1	1	1	1	1	1	1	1	1	1
	0.32	1.7	2.9	11	11	12	7.7	11	21	1.2×10^{15}
3	0.75	1	3.4	11	5.5	11	18	20	16	1.8×10^{15}
	1	2	4.2	13	7.9	11	18	17	37	3.4×10^2
	0.93	2.2	4.5	12	7.6	9.4	19	17	41	3.5×10^2
	0.12	0.51	0.06	0.23	0.23	0.055	0.031	0.0046	0.094	1
	1.4	3.8	0.64	1.9	18	19	57	0	41	11
	1	1	1	1	1	1	1	1	1	1
	0.69	18	3.9	13	42	18	8.9	16	38	2.2×10^{15}
4	1.4	2.6	7.8	9.1	9.2	19	20	35	34	8.1×10^{15}
	1.4	7.4	9.3	43	36	37	57	76	2.1×10^2	5.1×10^3
	1.4	9.9	11	43	39	33	66	1×10^2	2.5×10^2	5.5×10^3
	0.075	0.79	0.032	0.14	0.25	0.018	0.013	0	0.027	1
	1.4	5.8	5.1	2.4	83	79	2.2×10^2	0	8.5×10^2	12
	1	1	1	1	1	1	1	1	1	1
	2.3	54	4.2	20	1.2×10^2	24	12	18	83	3×10^{15}
5	2.1	4.6	9.3	33	14	25	19	1.6×10^2	63	1.6×10^{16}
	1.4	20	18	3×10^2	2.7×10^2	1.6×10^2	2.2×10^2	3.7×10^2	2.2×10^3	7.6×10^4
	1.5	31	25	3.6×10^2	2.9×10^2	1.5×10^2	2.6×10^2	5.3×10^2	2.8×10^3	1×10^5
	0.065	0.78	0.012	0.13	0.33	0.0066	0.0055	0	0.032	1

References

1. Nocedal, J.; Wright, S.J. *Numerical Optimization*; Springer: New York, NY, USA, 2006.
2. Boyd, S.; Vandenberghe, L. *Convex Optimization*; Cambridge University Press: Cambridge, UK, 2004.
3. Jordan, M.I.; Mitchell, T.M. Machine learning: Trends, perspectives, and prospects. *Science* **2015**, *349*, 255–260. [CrossRef] [PubMed]
4. Mayer-Schönberger, V.; Cukier, K. *Big Data: A Revolution That Will Transform How We Live, Work, and Think*; Houghton Mifflin Harcourt: Boston, MA, USA, 2013.
5. Zhang, W.; Gu, X.; Tang, L.; Yin, T.; Liu, D.; Zhang, Y. Application of machine learning, deep learning and optimization algorithms in geoengineering and geoscience: Comprehensive review and future challenge. *Gondwana Res.* **2022**, *109*, 1–17. [CrossRef]
6. Sun, S.; Cao, Z.; Zhu, H.; Zhao, J. A Survey of Optimization Methods From a Machine Learning Perspective. *IEEE Trans. Cybern.* **2019**, *50*, 3668–3681. [CrossRef] [PubMed]

7. Soydaner, D. A Comparison of Optimization Algorithms for Deep Learning. *Int. J. Pattern Recognit. Artif. Intell.* **2020**, *34*, 2052013. [CrossRef]
8. L'Heureux, A.; Grolinger, K.; Elyamany, H.F.; Capretz, M.A.M. Machine Learning with Big Data: Challenges and Approaches. *IEEE Access* **2017**, *5*, 7776–7797. [CrossRef]
9. Engelbrecht, A.P. *Computational Intelligence: An Introduction*; Wiley: Hoboken, NJ, USA, 2007.
10. Dennis, J.E.; Schnabel, R.B. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*; Siam: Bangkok, Thailand, 1996.
11. Broyden, C.G.; Fletcher, R.; Goldfarb, D.; Shanno, D.F. A class of quasi-Newton methods. *Math. Comput.* **1970**, *24*, 125–136. [CrossRef]
12. Nelder, J.A.; Mead, R. A simplex method for function minimization. *Comput. J.* **1965**, *7*, 308–313. [CrossRef]
13. Hooke, R.; Jeeves, T. Direct search solutions of numerical and statistical problems. *J. Assoc. Comput. Mach.* **1961**, *8*, 212–229. [CrossRef]
14. Dixon, L.C.W.; Szego, G.P. *Towards Global Optimization*; North-Holland: Amsterdam, The Netherlands, 1975.
15. Dixon, L.C.W.; Szego, G.P. *Towards Global Optimization II*; North-Holland: Amsterdam, The Netherlands, 1978.
16. Ali, M.M.; Storey, C. Topographical Multilevel Single Linkage. *J. Glob. Optim.* **1994**, *5*, 349–358. [CrossRef]
17. Wolfe, M.A. Interval methods for global optimization. *Appl. Math. Comput* **1996**, *75*, 179–206.
18. Allahdadi, M.; Nehi, H.M.; Ashayerinasab, H.A.; Javanmard, M. Improving the modified interval linear programming method by new techniques. *Inf. Sci.* **2016**, *339*, 224–236. [CrossRef]
19. Araya, I.; Reyes, V. Interval Branch-and-Bound algorithms for optimization and constraint satisfaction: A survey and prospects. *J. Glob. Optim.* **2016**, *65*, 837–866. [CrossRef]
20. Tsoulos, I.G.; Tzallas, A.; Tsalikakis, D. Use RBF as a Sampling Method in Multistart Global Optimization Method. *Signals* **2022**, *3*, 857–874. [CrossRef]
21. Rinnooy Kan, A.H.G.; Timmer, G.T. Stochastic Global Optimization Methods; Part I: Clustering Methods. *Math. Program.* **1987**, *39*, 27–56. [CrossRef]
22. Rinnooy Kan, A.H.G.; Timmer, G.T. Stochastic Global Optimization Methods; Part II: Clustering Methods. *Math. Program.* **1987**, *39*, 57–78. [CrossRef]
23. Torn, A.A. A search clustering approach to global optimization. In *Towards Global Optimization*; Dixon, L.C.W., Szego, G.P., Eds.; North-Holland: Amsterdam, The Netherlands, 1978; pp. 49–62.
24. Elwakeil, O.A.; Arora, J.S. Two algorithms for global optimization of general NLP problems. *Int. J. Numer. Methods Eng.* **1996**, *39*, 3305–3325. [CrossRef]
25. Sepulveda, A.E.; Epstein, L. The repulsion algorithm, a new multistart method for global optimization. *Struct. Optim.* **1996**, *11*, 145–152. [CrossRef]
26. Tsoulos, I.G.; Lagaris, I.E. MinFinder Locating all the local minima of a function. *Comput. Phys. Commun.* **2006**, *174*, 166–179. [CrossRef]
27. Day, R.F.; Yin, P.Y.; Wang, Y.C.; Chao, C.H. A new hybrid multi-start tabu search for finding hidden purchase decision strategies in WWW based on eye-movements. *Appl. Soft Comput.* **2016**, *48*, 217–229. [CrossRef]
28. Larson, J.; Wild, S.M. Asynchronously parallel optimization solver for finding multiple minima. *Math. Comput.* **2018**, *10*, 303–332. [CrossRef]
29. Jaiswal, P.; Larson, J. Multistart algorithm for identifying all optima of nonconvex stochastic functions. *Optim. Lett.* **2024**, *18*, 1335–1360. [CrossRef]
30. Devroye, L. *Non-Uniform Random Variate Generation*; Springer: New York, NY, USA, 1986.
31. Niederreiter, H. *Random Number Generation and Quasi-Monte Carlo Methods*; SIAM: Philadelphia, PA, USA, 1992.
32. Mockus, J. *Bayesian Approach to Global Optimization*; Springer: Dordrecht, The Netherlands, 1989.
33. McKay, M.D.; Beckman, R.J.; Conover, W.J. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics* **1979**, *21*, 239–245.
34. Wang, Z.; Zhao, D.; Heidari, A.A.; Chen, Y.; Chen, H.; Liang, G. Improved Latin hypercube sampling initialization-based whale optimization algorithm for COVID-19 X-ray multi-threshold image segmentation. *Sci. Rep.* **2024**, *14*, 13239. [CrossRef]
35. Borisut, P.; Nuchitprasittichai, A. Adaptive Latin Hypercube Sampling for a Surrogate-Based Optimization with Artificial Neural Network. *Processes* **2023**, *11*, 3232. [CrossRef]
36. Iordanis, I.; Koukouvinos, C.; Silou, I. Classification accuracy improvement using conditioned Latin Hypercube Sampling in Supervised Machine Learning. In Proceedings of the 2022 12th International Conference on Dependable Systems, Services and Technologies (DESSERT), Athens, Greece, 9–11 December 2022; pp. 1–5.
37. Rump, S.M. INTLAB—INTerval LABoratory. In *Developments in Reliable Computing*; Csendes, T., Ed.; Kluwer Academic Publishers: Dordrecht, The Netherlands, 1999; pp. 77–104.
38. Rump, S.M. Fast and Parallel Interval Arithmetic. *BIT* **1999**, *39*, 534–554. [CrossRef]

39. Song, C.; Kawai, R. Monte Carlo and variance reduction methods for structural reliability analysis: A comprehensive review. *Probabilistic Eng. Mech.* **2023**, *73*, 103479. [CrossRef]
40. Ramon, E. *Moore Interval Analysis*; Prentice-Hall: Englewood Cliffs, NJ, USA, 1966.
41. Kulisch, U.; Hammer, R.; Hocks, M.; Ratz, D. *C++ Toolbox for Verified Computing I*; Springer: Berlin/Heidelberg, Germany, 1995.
42. Nikas, I.A.; Grapsa, T.N. An escape-from-local minima technique in unconstrained optimization using a grid-like approach and interval equations. In Proceedings of the Numan2012 "5th Conference in Numerical Analysis", Ioannina, Greece, 5–8 September 2012.
43. Ratschek, H.; Rokne, J. *Computer Methods for the Range of Functions*; Ellis Horwood: Hemel Hempstead, UK, 1984.
44. Ratz, D. Inclusion Isotone Extended Interval Arithmetic. In *Institut für Angewandte Mathematik*; Universität Karlsruhe: Karlsruhe, Germany, 1996.
45. Moore, R.E.; Kearfott, R.B.; Cloud, M.J. *Introduction to Interval Analysis*; SIAM: Philadelphia, PA, USA, 2009.
46. Hansen, E.R.; Walster, G.W. *Global Optimization Using Interval Analysis*, 2nd ed.; Marcel Dekker: New York, NY, USA, 2004.
47. Neumaier, A. *Interval Methods for Systems of Equations*; Cambridge University Press: Cambridge, UK, 1990.
48. Alefeld, G.; Herzberger, J. *Introduction to Interval Computation*; Academic Press: New York, NY, USA, 1983.
49. Shahriari, B.; Swersky, K.; Wang, Z.; Adams, R.P.; de Freitas, N. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proc. IEEE* **2016**, *104*, 148–175. [CrossRef]
50. Bellman, R. *Dynamic Programming*; Princeton University Press: Princeton, NJ, USA, 1957.

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