applied sciences

Article

Pure Random Orthogonal Search (PROS): A Plain and Elegant Parameterless Algorithm for Global Optimization

Vagelis Plevris [1,2,*], Nikolaos P. Bakas [3] and German Solorzano [2]

1 Department of Civil and Architectural Engineering, Qatar University, Doha P.O. Box 2713, Qatar
2 Department of Civil Engineering and Energy Technology, OsloMet—Oslo Metropolitan University, Pilestredet 35, 0166 Oslo, Norway; germanso@oslomet.no
3 Computation-Based Science and Technology Research Center, The Cyprus Institute, 20 Konstantinou Kavafi Street, Nicosia 2121, Cyprus; n.bakas@cyi.ac.cy
* Correspondence: vplevris@qu.edu.qa; Tel.: +974-44034185

Featured Application: The proposed optimization method is a general tool which can have numerous applications in various mathematical, engineering and technological fields. Due to its simplicity and straight-forward implementation, it can easily be applied by non-experts to obtain optimized results.

Abstract: A new, fast, elegant, and simple stochastic optimization search method is proposed, which exhibits surprisingly good performance and robustness considering its simplicity. We name the algorithm pure random orthogonal search (PROS). The method does not use any assumptions, does not have any parameters to adjust, and uses basic calculations to evolve a single candidate solution. The idea is that a single decision variable is randomly changed at every iteration and the candidate solution is updated only when an improvement is observed; therefore, moving orthogonally towards the optimal solution. Due to its simplicity, PROS can be easily implemented with basic programming skills and any non-expert in optimization can use it to solve problems and start exploring the fascinating optimization world. In the present work, PROS is explained in detail and is used to optimize 12 multi-dimensional test functions with various levels of complexity. The performance is compared with the pure random search strategy and other three well-established algorithms: genetic algorithms (GA), particle swarm optimization (PSO), and differential evolution (DE). The results indicate that, despite its simplicity, the proposed PROS method exhibits very good performance with fast convergence rates and quick execution time. The method can serve as a simple alternative to established and more complex optimizers. Additionally, it could also be used as a benchmark for other metaheuristic optimization algorithms as one of the simplest, yet powerful, optimizers. The algorithm is provided with its full source code in MATLAB for anybody interested to use, test or explore.

Keywords: optimization; no free lunch; Occam's razor; orthogonal search; search problems; PROS

check for updates

Citation: Plevris, V.; Bakas, N.P.; Solorzano, G. Pure Random Orthogonal Search (PROS): A Plain and Elegant Parameterless Algorithm for Global Optimization. *Appl. Sci.* **2021**, *11*, 5053. https://doi.org/10.3390/app11115053

Academic Editor: Federico Divina

Received: 11 May 2021
Accepted: 26 May 2021
Published: 29 May 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 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

Optimization problems of all sorts arise in quantitative disciplines, ranging from computer science and engineering to operations research and economics. Thus, the development of solution methods has been of interest in mathematics for many centuries. The goal of optimization methods is to find an optimal (globally optimal) or near-optimal solution with low computational effort. We can distinguish between two different types of optimization methods: deterministic or exact methods that can guarantee finding the global optimal solution and heuristic methods where there is no such guarantee and a local optimal is usually obtained instead. The deterministic approach has a solid mathematical formulation, provides exact solutions, and can be used to solve optimization problems where the computational effort grows polynomially with respect to the problem size. The situation is different if the problem is NP-hard as the required computational effort grows exponentially. In this case, the optimization problems can become intractable or impractical to solve using deterministic methods. Heuristic optimization methods, on the other hand, can overcome such problems as they have shown good performance for many NP-complete problems and complex optimization tasks of practical relevance.

The terms heuristic or metaheuristic are commonly associated with random search algorithms. Random search algorithms refer to a set of optimization methods that implement some kind of randomness or probability (typically in the form of a pseudo-random number) in the definition of the iterative search strategy. This family of methods includes genetic algorithms (GA) [1–3], particle swarm optimization (PSO) [4–6], differential evolution (DE) [7–10], ant colony optimization (ACO) [4,11], harmony search (HS) [12,13], simulated annealing [14], tabu search [15], and many others.

The introduced randomness in random search algorithms possesses several advantages that add robustness to the solution strategy. It can significantly increase the solution's convergence speed and make the method less sensitive to modeling errors. Furthermore, it may enable the method to escape a local optimum and push the search towards the global solution. Therefore, random search algorithms are very useful for ill-structured global optimization problems where the objective function may be nonconvex, nondifferentiable, and possibly, discontinuous over a continuous, discrete, or mixed continuous-discrete domain. Nevertheless, they offer no guarantee that a global optimal solution will be obtained.

Although numerous heuristic optimization methods have been proposed thus far in the literature, surprisingly, new methods are still being proposed almost every day. These new methods may differ from existing ones, but the major question is whether they make a real difference or a significant improvement. Often, a newly proposed optimization algorithm is simply a replica of an existing method with a few extra parameters or rules that are supposed to improve or "optimize" its performance. However, adding more complexity is not always a good idea as the fine-tuning of the newly introduced parameters ends up being another optimization problem by itself.

In the present paper, we introduce a new optimization method: pure random orthogonal search (PROS), which is simple, elegant, fast, easy to understand and implement, and truly parameterless. The performance of the new proposed strategy is investigated in comparison to pure random search (PRS) as well as three other established and well-known optimization methods, namely genetic algorithm (GA), particle swarm optimization (PSO), and differential evolution (DE). Due to the simplicity and the lack of parameters of PROS, this method can set a benchmark for other optimization algorithms that have a higher degree of complexity and require the definition of one or more parameters.

The remainder of the paper is organized as follows. The optimization concept, the no free lunch theorem in optimization and the concept of simplicity are presented in Sections 1.1 and 1.2. The formulation of the optimization problem is presented in Section 2.1. Section 2.2 describes the simple PRS approach while in Section 2.3 there is a description of the proposed PROS optimization algorithm. Section 2.4 describes the methodology and the objective functions used in the study. The results are presented in Section 3 followed by a discussion and the conclusions. Appendix A contains the acronyms, notation, and symbols used in the study. Appendix B provides further detailed information about the 12 objective functions considered.

1.1. No Free Lunch Theorem in Optimization

In optimization, the no free lunch (NFL) theorem [16,17] is an impossibility theorem stating that, for certain types of mathematical problems, the computational cost of finding a solution, averaged over all problems in the class, is the same for any solution method. In other words, some methods tend to work well in some problems, and other methods work well in other problems, but in general one cannot say that method A is better than B when all problems are considered altogether. Wolpert and Macready [16] demonstrate the danger of comparing algorithms by their performance on a small sample of problems and they indicate the importance of incorporating problem-specific knowledge into the behavior of the algorithm. They proved that for all possible domains (all possible problem instances drawn from a uniform probability distribution), the average performance for two algorithms A and B is the same.

A conventional, but not entirely accurate, interpretation of NFL is that "a general-purpose universal optimization strategy is theoretically impossible, and the only way one strategy can outperform another is if it is specialized to the specific problem under consideration" [18]. In other words, "no search algorithm, no matter how sophisticated, should a priori be expected to give better performance than any other". If no prior assumptions about the optimization problem can be made, no optimization strategy can be expected to perform better than any other strategy (including pure random searching). A general-purpose universal optimization strategy is theoretically impossible, and the only way one strategy can outperform another is to be specialized for the specific problem at hand.

This is in full accordance with the enormous number of optimization methods proposed in the literature every year. Tens or hundreds of new heuristic or other methods are proposed by researchers, either inspired by nature or by human imagination, and it is extremely difficult to try them all or even simply keep track of them. It is obvious that if a single method were globally and clearly better than the alternatives, without any doubt, gradually all researchers would adopt it to get better results and the other methods would be soon completely abandoned. Merely the fact that so many methods are still being developed and used today proves that there is no overall "best" method for all problems and all cases. The method one uses appears to be a matter of personal preference and taste, rather than a decision based on scientific findings and facts, although many researchers still claim that their own method is "the best".

1.2. Occam's Razor and Simplicity in Optimization

Occam's razor is a problem-solving principle stating that "other things being equal, simpler explanations are generally better than more complex ones", an idea originally attributed to William of Ockham, an English Franciscan philosopher who lived in the 14th century. He used a preference for simplicity to defend the idea of divine miracles. In other words, "The simplest solution is most likely the right one if results match". Alternatively, "When presented with competing hypotheses that make the same predictions, one should select the solution with the fewest assumptions". In science, Occam's razor is used as an abductive heuristic in the development of theoretical models. Simpler theories are in general preferable to more complex ones because they are more testable and easier for people to understand, remember and use.

How does that apply to optimization algorithms? We will investigate this through an example. Expert A proposes an elegant, simple and clever optimization algorithm OA. Then researcher B in an effort to make a contribution of his/her own, takes this algorithm and proposes a new version that is usually named as "Improved OA" (IOA) or similarly "Enhanced AO", "Better AO", "Modified AO", etc. To make this "improvement", B has no hesitation to add a number of extra parameters to the original algorithm, essentially adding extra complexity to it. In the work published by B, it is "proven" that IOA works better than the original OA in the problems examined. However, in the end, only rarely is IOA acknowledged and recognized as generally better than OA by the scientific community as a whole. Why is that? Assuming that B is not trying to cheat and truly believes in the superiority of IOA over OA, there are three main reasons that this can happen:

- **Test problems bias**. IOA has better performance than OA in the test examples published by B, but not in other problems. In other words, there is some bias as the problems chosen and presented are in favor of IOA in comparison to OA.
- **Parameter bias**. The extra parameters introduced in IOA work good for some problems, such as the ones chosen and presented by B, but not in others. And of course, extra knowledge and experience is needed in order to find the right values of the extra parameters for every single problem.

- **Author bias**. A combination of the two above-mentioned points. Author B knows the IOA better than anyone else and has also experience in optimization problems. As a result, B can fine-tune the algorithm in such a way that he/she can achieve optimal performance, but the same is not true for other users of the algorithm facing different optimization problems. Thus, the "improved" algorithm works good only if it can be fine-tuned by an expert, such as B.

In some extreme cases, we have seen authors adding so much complexity and so many extra parameters to an algorithm that literally a new optimization problem needs to be solved in order to fine-tune the parameters of the optimization method to achieve good results. As such, the question is always if it is worth it to add complexity to a method that simply works well. In the opinion of the authors, adding some complexity is not bad in itself, but it should always be justified by managing to achieve clearly better results in the majority of the problems, while also keeping things clear, simple and elegant.

2. Materials and Methods

2.1. Formulation of the Optimization Problem

A typical optimization problem without additional constraints (other than the decision variables bounds) can be formulated as follows:

$$\min_{x \in \Omega} f(\boldsymbol{x}) \tag{1}$$

where $\boldsymbol{x}$ is a vector of decision variables, $\boldsymbol{x} = \{x_1, \ldots, x_D\}$ (candidate solution in the search space), $f\colon \mathbb{R}^D \to \mathbb{R}$ is a scalar-valued function called objective (or cost) function to be minimized, while D is the dimensionality of the optimization problem. In the present work, the search space $\Omega \subset \mathbb{R}^D$ is defined by the lower bound and upper bound vectors as $\Omega = [\boldsymbol{lb}, \boldsymbol{ub}]$, i.e., $lb_i \leq x_i \leq ub_i$ for every $i \in \{1, \ldots, D\}$. Truly unconstrained optimization corresponds to the case where $\Omega = \mathbb{R}^D$.

In this work, we focus on problems without constraints. Real life problems rarely have to do with unconstrained optimization as they usually involve several constraints. There are various methods that can be used for handling constrained problems with heuristic algorithms, such as methods based on preserving the feasibility of the solutions, methods based on penalty functions, and others [19,20]. Similar constraint handling techniques can be also easily used with the proposed PROS method.

2.2. Pure Random Search

Pure random search [21] (PRS) can be considered as the simplest and most obvious random search method, also known as "blind search" [22]. The method, first defined by Brooks [23], simply searches for candidate solutions at random within the search space by sampling repeatedly from the feasible region Ω, typically according to a uniform sampling distribution. The method in its simplest form maintains and generates a single point at each iteration and it does not adapt the current sampling strategy to information that has been previously gathered during the search. It can be proven that PRS converges on the global optimum with probability one [22], yet the convergence speed is very slow. In addition, the expected number of function evaluations for PRS to find the optimal solution within a given value (ε) increases exponentially with the dimensionality (D) of the problem, which makes it useless in problems with high dimensionality. The method has major difficulties navigating towards an optimum solution, especially in search spaces with a high number of dimensions. Improvements of the basic method have been suggested [24] together with more sophisticated versions of it, such as pure adaptive search [21], where each iteration "adapts" to the current level set by restricting its search domain to improving points, or localized random search [21].

The single-point PRS algorithm can be described as follows:

- Initialize $\boldsymbol{x}$ with a random position in the search space (feasible region Ω), i.e., $lb_\mathrm{i} \leq x_\mathrm{i} \leq ub_\mathrm{i}$ for every $i \in \{1, \ldots, D\}$

- Until a termination criterion is met (e.g., number of iterations performed, or adequate fitness reached), repeat the following:
  - Sample a new position $\boldsymbol{y}$ randomly in the search space, using uniform distribution
  - If $f(\boldsymbol{y}) < f(\boldsymbol{x})$ then move to the new position by setting $\boldsymbol{x} = \boldsymbol{y}$
- At the end, $\boldsymbol{x}$ holds the best-found position and $f(\boldsymbol{x})$ is the best objective function value found.

2.3. Pure Random Orthogonal Search

The proposed method, pure random orthogonal search (PROS) is similar to PRS, but instead of searching in the search space with a new solution that is completely different and independent from the previous solution (i.e., all D positions of the new vector of decision variables are changed in each iteration), it changes only one variable of the current solution at a time. In other words, in each iteration, the new candidate solution is different from the previous solution in only one design variable (one dimension), randomly.

The PROS algorithm can be described as follows:

- Initialize $\boldsymbol{x}$ with a random position in the search space (feasible region Ω), i.e., $lb_{\mathrm{i}} \leq x_{\mathrm{i}} \leq ub_{\mathrm{i}}$ for every $i \in \{1, \ldots, D\}$
- Until a termination criterion is met, repeat the following:
  - Set $\boldsymbol{y} = \boldsymbol{x}$
  - Sample a random integer $j \in \{1, \ldots, D\}$
  - Sample a random number r in the range $[lb_j, ub_j]$ using uniform distribution
  - Set $y_j = r$
  - If $f(\boldsymbol{y}) < f(\boldsymbol{x})$ then move to the new position by setting $\boldsymbol{x} = \boldsymbol{y}$
- At the end, $\boldsymbol{x}$ holds the best-found position and $f(\boldsymbol{x})$ is the best solution.

Figure 1 shows a flowchart of the algorithm. It has to be noted that PROS is not a population-based method as it works with a single solution during the search. The same solution is updated, one variable at a time, until a termination criterion is met. The method is simple, elegant, fast, and easy to understand and implement in any computer language.

Figure 2 shows the search path of the PROS algorithm for two examples in 2D. The functions F01-Sphere function (sphere_func) in $[-100, 100]^2$ and F10-Rastrigin's function (rastrigin_func) in $[-5.12, 5.12]^2$ have been used as example cases. The functions are described in detail in Appendix B, together with the other functions used in the study.

Table 1 shows the first five steps of the PROS algorithm in an example case where the F01-Sphere function (sphere_func) in five dimensions is optimized in the region $[-100, 100]^5$. For each candidate solution $\boldsymbol{y}_{\mathrm{i}}$ at step i, the bold cell in brackets ([]) represents the j-th element that is different than the corresponding element of the current solution $\boldsymbol{x}_{\mathrm{i}}$.

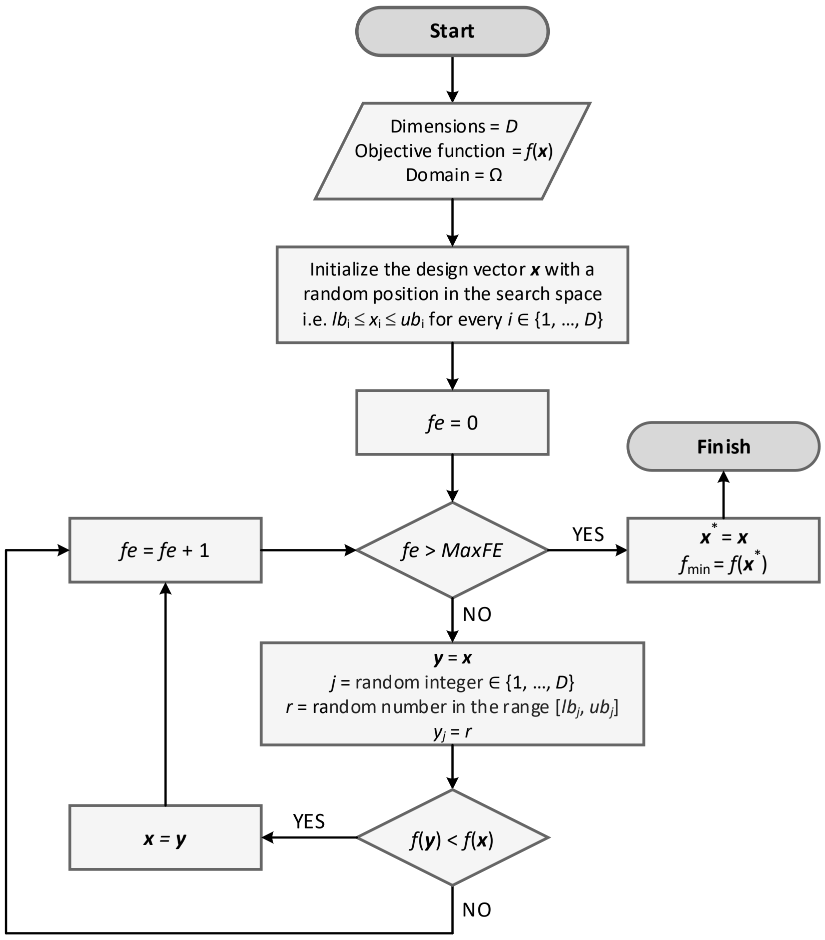

Start
Dimensions = *D*
Objective function = *f*(***x***)
Domain = Ω
Initialize the design vector ***x*** with a random position in the search space i.e. $lb_i \le x_i \le ub_i$ for every $i \in \{1, ..., D\}$
fe = 0
fe > *MaxFE*
YES
$\boldsymbol{x}^* = \boldsymbol{x}$
$f_{min} = f(\boldsymbol{x}^*)$
Finish
NO
y = ***x***
j = random integer ∈ {1, ..., *D*}
r = random number in the range [lb_j, ub_j]
$y_j = r$
f(***y***) < *f*(***x***)
YES
x = ***y***
NO
fe = *fe* + 1

Figure 1. Flowchart of the PROS algorithm.

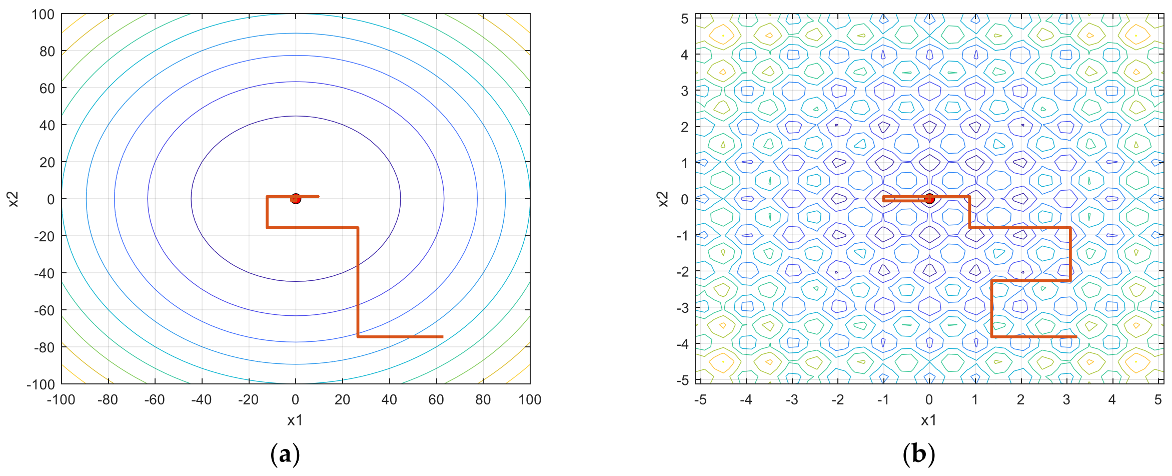


Figure 2. Search path of the PROS algorithm in two 2D problem examples: (**a**) F01-Sphere function (sphere_func) in $[-100, 100]^2$; (**b**) F10-Rastrigin's function (rastrigin_func) in $[-5.12, 5.12]^2$.

Table 1. Example case: the 5 first steps of the PROS algorithm for the F01-Sphere function (sphere_func) in [–100, 100]^2.

Step	x_i/y_i	Design Vector					Obj. Value	Notes
		1	2	3	4	5		
1	x_1	**−34.32**	−5.13	−58.68	1.29	−22.26	$5.15 \cdot 10^3$	No improvement
	y_1 (j = 4)	−34.32	−5.13	−58.68	**[14.73]**	−22.26	$\mathbf{5.36 \cdot 10^3}$	keep x_1 for Step 2 ($x_2 = x_1$)
2	x_2	−34.32	−5.13	−58.68	1.29	−22.26	$5.15 \cdot 10^3$	Improvement
	y_2 (j = 3)	−34.32	−5.13	**[35.52]**	1.29	−22.26	$\mathbf{2.96 \cdot 10^3}$	keep y_2 for Step 3 ($x_3 = y_2$)
3	x_3	−34.32	−5.13	35.52	1.29	−22.26	$2.96 \cdot 10^3$	No improvement
	y_3 (j = 5)	−34.32	−5.13	35.52	1.29	**[−43.44]**	$\mathbf{4.36 \cdot 10^3}$	keep x_3 for Step 4 ($x_4 = x_3$)
4	x_4	−34.32	−5.13	35.52	1.29	−22.26	$2.96 \cdot 10^3$	Improvement
	y_4 (j = 1)	**[11.97]**	−5.13	35.52	1.29	−22.26	$\mathbf{1.93 \cdot 10^3}$	keep y_4 for Step 5 (x_5=y_4)
5	x_5	11.97	−5.13	35.52	1.29	−22.26	$1.93 \cdot 10^3$	No improvement
	y_5 (j = 5)	11.97	−5.13	35.52	1.29	**[−88.95]**	$\mathbf{9.35 \cdot 10^3}$	keep x_5 for Step 6 (x_6=x_5)

2.4. *Methodology of the Study and Objective Functions Used*

We compare the PROS algorithm with the simplistic PRS approach and three other well-known algorithms, namely GA [1–3], PSO [4–6], and DE [7–10] in 12 mathematical test functions. All test functions are multi-dimensional and can be defined with any number of dimensions. We use a set of four different numbers of dimensions (D) in our study: (i) D = 5, (ii) D = 10, (iii) D = 30, (iv) D = 50. When D = 5 is used, the design vector of each problem has 5 elements, i.e., $\boldsymbol{x} = [x_1, x_2, x_3, x_4, x_5]$ and the optimization problem is relatively simple. This is not the case with higher values of D where the problem gets significantly harder. For example, in the case D = 50, each design vector has 50 elements to adjust (x_1 to x_{50}). The five algorithms used are the following:

1. Genetic algorithm (GA);
2. Particle swarm optimization (PSO);
3. Differential evolution (DE);
4. Pure random search (PRS);
5. Pure random orthogonal search (PROS).

PRS and PROS use a single point for their search, while GA, PSO, and DE use a population of agents (or particles) at each generation (iteration). In order to compare the algorithms fairly we will use the maximum function evaluations as the convergence criterion for all cases, i.e., all algorithms will stop after a certain number of function evaluations is completed. GA and PSO are based on their MATLAB implementations and are executed using the MATLAB commands *ga* and *particleswarm*, respectively, using their default parameters. DE, PRS, and PROS were programmed in MATLAB from scratch. For DE, a DE/rand/1/bin scheme is used with differential weight F = 0.6 and crossover probability CR = 0.8.

Each optimization problem is run 10 times. The total number of optimization problems solved is 5 (methods) * 4 (different dimensions) * 12 (Problems) * 10 (Runs) = 2400. To maintain a consistency for all problems and all the different cases, in population-based algorithms (GA, PSO, DE), the population size is set to $NP = 10 \cdot D$ and the maximum number of iterations (or generations) is set to $MaxIter = 20 \cdot D - 50$. Although some specific problems would actually require more iterations, we decided not to make exceptions to this rule, to keep things consistent and we kept these values that work well for most problems. Then the max. number of function evaluations can be calculated as $MaxFE = NP \cdot MaxIter$. Table 2 shows the population size, max. number of generations/iterations and the max. number of function evaluations for each category of problems, based on the number of dimensions. For example, when we investigate the performance of the five algorithms in 30 dimensions (D = 30), the population size used in the three population-based methods (GA, PSO and DE) is 300, while the maximum number of generations (or iterations) for population-based methods is set to 550. Equivalently, for the two methods that are not population-based (PRS and PROS), the convergence criterion has to do with the maximum number of objective function evaluations *MaxFE* which is set to 165,000.

Table 2. Optimization parameters and convergence criteria used for each category of problems based on the number of dimensions.

No of Dimensions, D	D = 5	D = 10	D = 30	D = 50
Population size NP $NP = 10 \cdot D$	50	100	300	500
Max. Generations/Iterations $MaxIter$ $MaxIter = 20 \cdot D - 50$	50	150	550	950
Max. Obj. function evaluations $MaxFE$ $MaxFE = NP \cdot MaxIter$	2500	15,000	165,000	475,000

2.5. Objective Functions

We used 12 mathematical objective functions of different levels of complexity in our study. All functions are defined in multiple dimensions and are to be minimized, with the minimum value being zero. The objective functions together with their search range, the optimum value $f(\boldsymbol{x}^*)$, the location of the optimum $\boldsymbol{x}^*$ in the design space and some other properties are presented in Table 3. More details on the functions can be found in Appendix B, where the functions are plotted for the two-dimensional case (D = 2). The aim of every optimization algorithm used in the study is to find the design vector $\boldsymbol{x}^*$ that minimizes the value of f, for each function f_{i} (I = 1,2, . . . ,12).

Table 3. The 12 objective functions used in the study.

ID	Name and Code Name	Search Range	Minimum	Properties
F01	Sphere function sphere_func	$[-100, 100]^D$	$f_{01}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{0, 0, \ldots, 0\}$	Unimodal Highly symmetric, in particular rotationally invariant
F02	Ellipsoid function ellipsoid_func	$[-100, 100]^D$	$f_{02}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{0, 0, \ldots, 0\}$	Unimodal Symmetric
F03	Sum of Different Powers function sumpow_func	$[-10, 10]^D$	$f_{03}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{0, 0, \ldots, 0\}$	Unimodal
F04	Quintic function quintic_func	$[-20, 20]^D$	$f_{04}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{-1, -1, \ldots, -1\}$ or $\boldsymbol{x}^* = \{2, 2, \ldots, 2\}$	Has two global optima
F05	Drop-Wave function drop_wave_func	$[-5.12, 5.12]^D$	$f_{05}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{0, 0, \ldots, 0\}$	Multi-modal Highly complex
F06	Weierstrass function weierstrass_func	$[-0.5, 0.5]^D$	$f_{06}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{0, 0, \ldots, 0\}$	Multi-modal Continuous everywhere but only differentiable on a set of points
F07	Alpine 1 function alpine1_func	$[-10, 10]^D$	$f_{07}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{0, 0, \ldots, 0\}$	Multi-modal
F08	Ackley's function ackley_func	$[-32.768, 32.768]^D$	$f_{08}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{0, 0, \ldots, 0\}$	Multi-modal Having many local optima with the global optima located in a very small basin
F09	Griewank's function griewank_func	$[-100, 100]^D$	$f_{09}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{0, 0, \ldots, 0\}$	Multi-modal With many regularly distributed local optima
F10	Rastrigin's function rastrigin_func	$[-5.12, 5.12]^D$	$f_{10}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{0, 0, \ldots, 0\}$	Multi-modal With many regularly distributed local optima
F11	HappyCat function happycat_func	$[-20, 20]^D$	$f_{11}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{-1, -1, \ldots, -1\}$	Multi-modal Global optimum located in curved narrow valley
F12	HGBat function hgbat_func	$[-15, 15]^D$	$f_{12}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{-1, -1, \ldots, -1\}$	Multi-modal Global optima located in curved narrow valley

The 12 functions exhibit different computational complexities. Some are very easy to calculate, while others are much harder. Figure 3 shows the computational time needed for the calculation of the values of the objective functions 100,000 times, for a vector of 5, 10, 30 or 50 variables (D = 5, 10, 30, 50). All calculations were done in MATLAB with a 64-bit computer running Windows 10, equipped with an Intel i9-8950HK CPU @ 2.90GHz and 32 GB RAM. In the D = 5 and D = 10 cases, most functions take 0.1 to 0.7 s to calculate. The exception is function F06 which is much harder and requires 5.9 s and 11 s to calculate, for D = 5 and D = 10, respectively. Note that the vertical axis of each diagram is in logarithmic scale, for better clarity of the results. It appears that some functions become much harder to calculate as the dimensionality increases (for example functions F03, F04, F06, partly F08) while for others this is not the case.

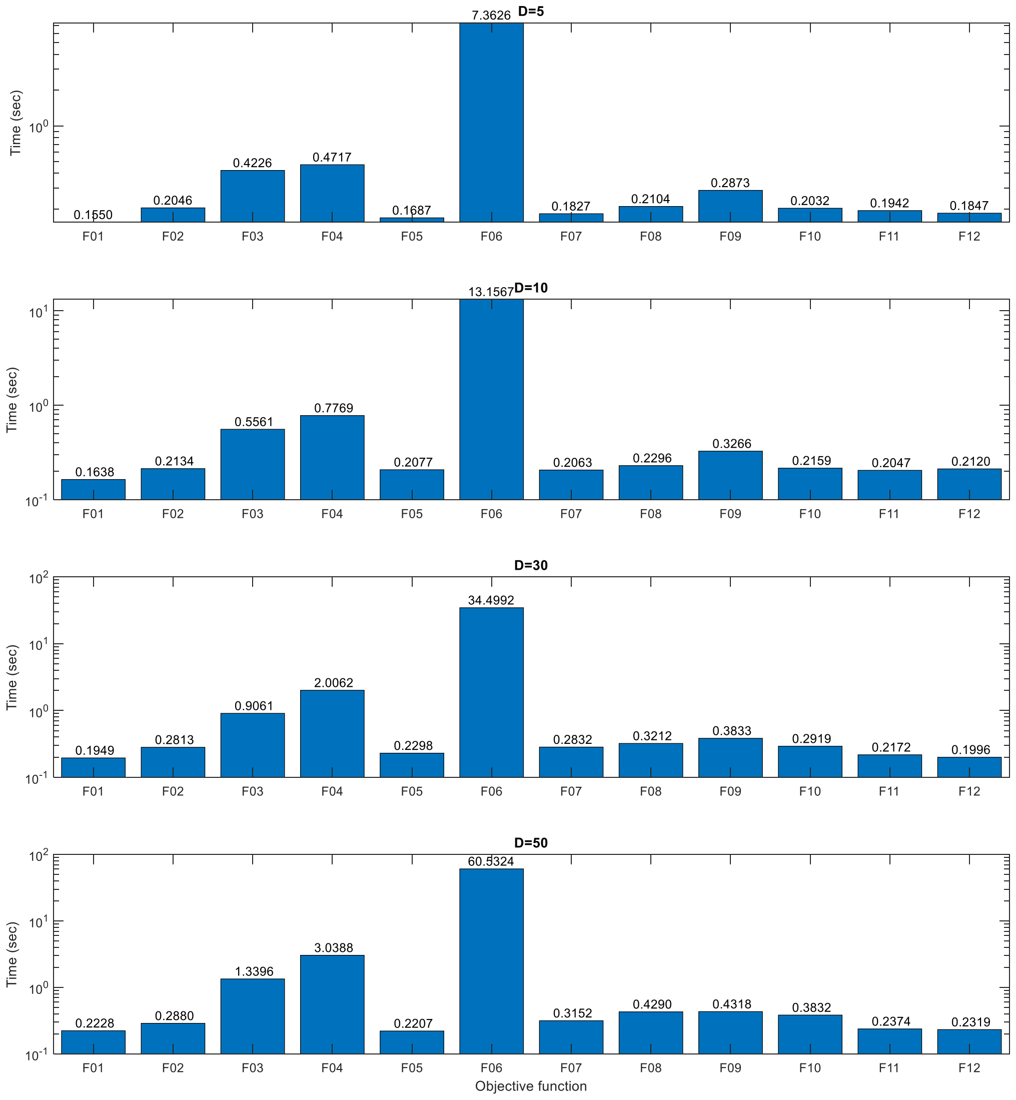


Figure 3. Time needed for the calculation of each objective function 100,000 times, for a random vector of 5, 10, 30 or 50 variables (D = 5, 10, 30, 50). The y-axis is in logarithmic scale.

3. Results

3.1. Objective Function Values

In all 12 test examples, the minimum (target) value of the objective function is zero, as shown in Table 3. The results of the PROS algorithm in terms of minimum objective value achieved, are presented in Table 4 (as the average of 10 runs).

Table 4. Best results * of the PROS algorithm for all 12 objective functions, for *D* = 5, 10, 30, 50 dimensions (averages of 10 runs). The target (optimum) value is zero in all the cases considered.

ID	Function Name	*D* = 5	*D* = 10	*D* = 30	*D* = 50
F01	Sphere	4.99E-01	9.56E-02	2.17E-02	9.25E-03
F02	Ellipsoid	9.78E-01	6.72E-01	2.92E-01	3.77E-01
F03	Sum of Dif. Powers	8.07E-04	5.58E-05	5.35E-06	2.80E-06
F04	Quintic	8.56E-01	6.61E-01	5.40E-01	5.18E-01
F05	Drop-Wave	2.63E-01	5.48E-01	9.10E-01	9.74E-01
F06	Weierstrass	3.66E-01	3.62E-01	4.25E-01	5.36E-01
F07	Alpine 1	4.99E-03	3.16E-03	2.36E-03	2.46E-03
F08	Ackley's	7.92E-01	1.72E-01	3.99E-02	1.87E-02
F09	Griewank's	5.34E-02	4.13E-02	2.51E-02	1.13E-02
F10	Rastrigin's	3.44E-01	4.94E-02	1.03E-02	5.74E-03
F11	HappyCat	4.71E-01	4.25E-01	5.63E-01	6.92E-01
F12	HGBat	4.40E-01	4.61E-01	6.51E-01	5.97E-01

* Average value (in 10 runs) of the obj. function at the end of the optimization process.

The corresponding results of the other four algorithms, together with PROS, for number of dimensions *D* = 5, *D* = 10, *D* = 30, and *D* = 50 are presented in Table 5, Table 6, Table 7, and Table 8, respectively. Final values greater than one (i.e., not excellent solutions) are highlighted with bold font in the tables. Although it can be proved that the PRS algorithm will converge to the optimum solution at the end, it is obvious that convergence speed is very slow and, as a result the PRS algorithm, fails to give good solutions in almost all problems examined, given the allowed number of function evaluations. All PRS values are bold in the tables with the exception of the ones for problem F05. GA and PSO show good performance in general with the exception of a few cases, while the performance of DE is not so good, especially for higher numbers of dimensions (*D* = 30 or *D* = 50).

Table 5. Best results * for all 5 optimizers and all 12 objective functions, for 5 dimensions (*D* = 5, averages of 10 runs). The target (optimum) value is zero in all cases considered.

ID	Function Name	GA	PSO	DE	PRS	PROS
F01	Sphere	6.99E-04	9.04E-04	2.34E-01	**8.30E+02**	4.99E-01
F02	Ellipsoid	2.12E-03	5.58E-03	5.26E-01	**1.56E+03**	9.78E-01
F03	Sum of Dif. Powers	6.38E-06	1.85E-08	3.11E-05	**2.15E+01**	8.07E-04
F04	Quintic	2.49E-01	1.95E-01	**4.03E+00**	**2.33E+03**	8.56E-01
F05	Drop-Wave	1.56E-01	6.38E-02	9.55E-02	4.20E-01	2.63E-01
F06	Weierstrass	2.21E-01	4.50E-02	5.27E-01	**3.87E+00**	3.66E-01
F07	Alpine 1	6.18E-03	2.39E-03	3.65E-01	**1.93E+00**	4.99E-03
F08	Ackley's	1.89E-01	9.54E-02	6.52E-01	**1.19E+01**	7.92E-01
F09	Griewank's	1.96E-02	1.46E-01	2.60E-01	**1.00E+00**	5.34E-02
F10	Rastrigin's	5.20E-01	**3.70E+00**	**7.59E+00**	**2.08E+01**	3.44E-01
F11	HappyCat	5.31E-01	1.92E-01	3.89E-01	**5.41E+00**	4.71E-01
F12	HGBat	5.46E-01	2.40E-01	2.97E-01	**1.81E+01**	4.40E-01

* Average value (in 10 runs) of the obj. function at the end of the optimization process, for each algorithm.

Table 6. Best results * for all 5 optimizers and all 12 objective functions, for 10 dimensions (*D* = 10, averages of 10 runs). The target (optimum) value is zero in all cases considered.

ID	Function Name	GA	PSO	DE	PRS	PROS
F01	Sphere	6.93E-06	4.41E-11	1.02E-01	**4.83E+03**	9.56E-02
F02	Ellipsoid	5.89E-05	3.83E-10	4.03E-01	**1.83E+04**	6.72E-01
F03	Sum of Dif. Powers	4.22E-08	7.75E-23	1.57E-06	**2.04E+03**	5.58E-05
F04	Quintic	2.42E-02	3.52E-05	**1.04E+01**	**4.95E+04**	6.61E-01
F05	Drop-Wave	3.45E-01	1.24E-01	1.73E-01	7.85E-01	5.48E-01
F06	Weierstrass	1.24E-01	9.90E-05	8.30E-01	**1.03E+01**	3.62E-01
F07	Alpine 1	1.10E-03	4.56E-07	**1.68E+00**	**7.33E+00**	3.16E-03
F08	Ackley's	1.94E-03	1.98E-06	2.66E-01	**1.70E+01**	1.72E-01
F09	Griewank's	1.06E-02	1.48E-01	4.83E-01	**2.11E+00**	4.13E-02
F10	Rastrigin's	1.96E-04	**8.12E+00**	**3.61E+01**	**6.62E+01**	4.94E-02
F11	HappyCat	7.92E-01	1.86E-01	3.98E-01	**1.14E+01**	4.25E-01
F12	HGBat	8.00E-01	3.00E-01	3.09E-01	**1.02E+02**	4.61E-01

* Average value (in 10 runs) of the obj. function at the end of the optimization process, for each algorithm.

Table 7. Best results * for all 5 optimizers and all 12 objective functions, for 30 dimensions (*D* = 30, averages of 10 runs). The target (optimum) value is zero in all cases considered.

ID	Function Name	GA	PSO	DE	PRS	PROS
F01	Sphere	7.50E-07	4.03E-26	**3.70E+02**	**3.58E+04**	2.17E-02
F02	Ellipsoid	2.25E-05	1.15E-24	**2.80E+03**	**4.57E+05**	2.92E-01
F03	Sum of Dif. Powers	1.50E-12	1.54E-52	**3.50E+03**	**2.46E+15**	5.35E-06
F04	Quintic	1.80E-02	4.43E-14	**2.73E+03**	**1.92E+06**	5.40E-01
F05	Drop-Wave	7.29E-01	3.76E-01	8.71E-01	9.63E-01	9.10E-01
F06	Weierstrass	3.63E-01	1.55E-01	**2.06E+01**	**4.06E+01**	4.25E-01
F07	Alpine 1	3.37E-04	7.42E-15	**2.54E+01**	**3.86E+01**	2.36E-03
F08	Ackley's	6.89E-04	4.37E-14	**6.08E+00**	**1.94E+01**	3.99E-02
F09	Griewank's	1.23E-03	6.39E-03	**1.07E+00**	**1.01E+01**	2.51E-02
F10	Rastrigin's	1.99E-01	**4.56E+01**	**2.34E+02**	**3.14E+02**	1.03E-02
F11	HappyCat	9.79E-01	3.39E-01	9.38E-01	**2.88E+01**	5.63E-01
F12	HGBat	9.55E-01	4.95E-01	**7.12E+00**	**8.32E+02**	6.51E-01

* Average value (in 10 runs) of the obj. function at the end of the optimization process, for each algorithm.

The PROS algorithm shows a very good performance in the test functions examined. It clearly outperforms PRS which is very poor and although PROS is so simple in its formulation, the algorithm gives very competitive and even better results in comparison to the other established algorithms, i.e., GA, PSO and DE. In many cases, PROS outperforms some, or all, of the other algorithms. Surprisingly, the algorithm shows very good performance (compared to the others) in difficult cases with a large number of dimensions (*D* = 30 or *D* = 50).

As expected, PRS exhibits the poorest performance with very large values of the objective function in some cases. Figure 4 shows the best results (final values of the objective values) for each algorithm, for the different cases (as averages of 10 runs). PRS has been excluded from this comparison because its final objective function values are very large, and it would distort the diagram significantly. For better presentation of the results, the y-axis has been limited to the value of 1 in all cases. Since the target (optimum) value of each of the 12 functions used is zero, the lower bar height is better in this comparison.

Table 8. Best results * for all 5 optimizers and all 12 objective functions, for 50 dimensions ($D = 50$, averages of 10 runs). The target (optimum) value is zero in all cases considered.

ID	Function Name	GA	PSO	DE	PRS	PROS
F01	Sphere	1.03E-06	2.04E-34	**1.03E+04**	**8.00E+04**	9.25E-03
F02	Ellipsoid	3.50E-05	1.19E-32	**1.46E+05**	**1.77E+06**	3.77E-01
F03	Sum of Dif. Powers	8.98E-13	3.65E-60	**6.01E+14**	**1.50E+30**	2.80E-06
F04	Quintic	1.87E-02	3.42E-15	**4.79E+05**	**5.92E+06**	5.18E-01
F05	Drop-Wave	8.36E-01	6.09E-01	9.68E-01	9.82E-01	9.74E-01
F06	Weierstrass	4.66E-01	**2.08E+00**	**5.39E+01**	**7.30E+01**	5.36E-01
F07	Alpine 1	3.84E-04	1.99E-14	**6.33E+01**	**7.97E+01**	2.46E-03
F08	Ackley's	5.50E-04	1.16E-01	**1.53E+01**	**2.01E+01**	1.87E-02
F09	Griewank's	4.44E-08	2.47E-03	**3.59E+00**	**2.04E+01**	1.13E-02
F10	Rastrigin's	1.25E-04	**9.60E+01**	**5.02E+02**	**5.99E+02**	5.74E-03
F11	HappyCat	**1.08E+00**	5.01E-01	**8.57E+00**	**3.89E+01**	6.92E-01
F12	HGBat	**1.06E+00**	6.04E-01	**2.46E+02**	**1.75E+03**	5.97E-01

* Average value (in 10 runs) of the obj. function at the end of the optimization process, for each algorithm.

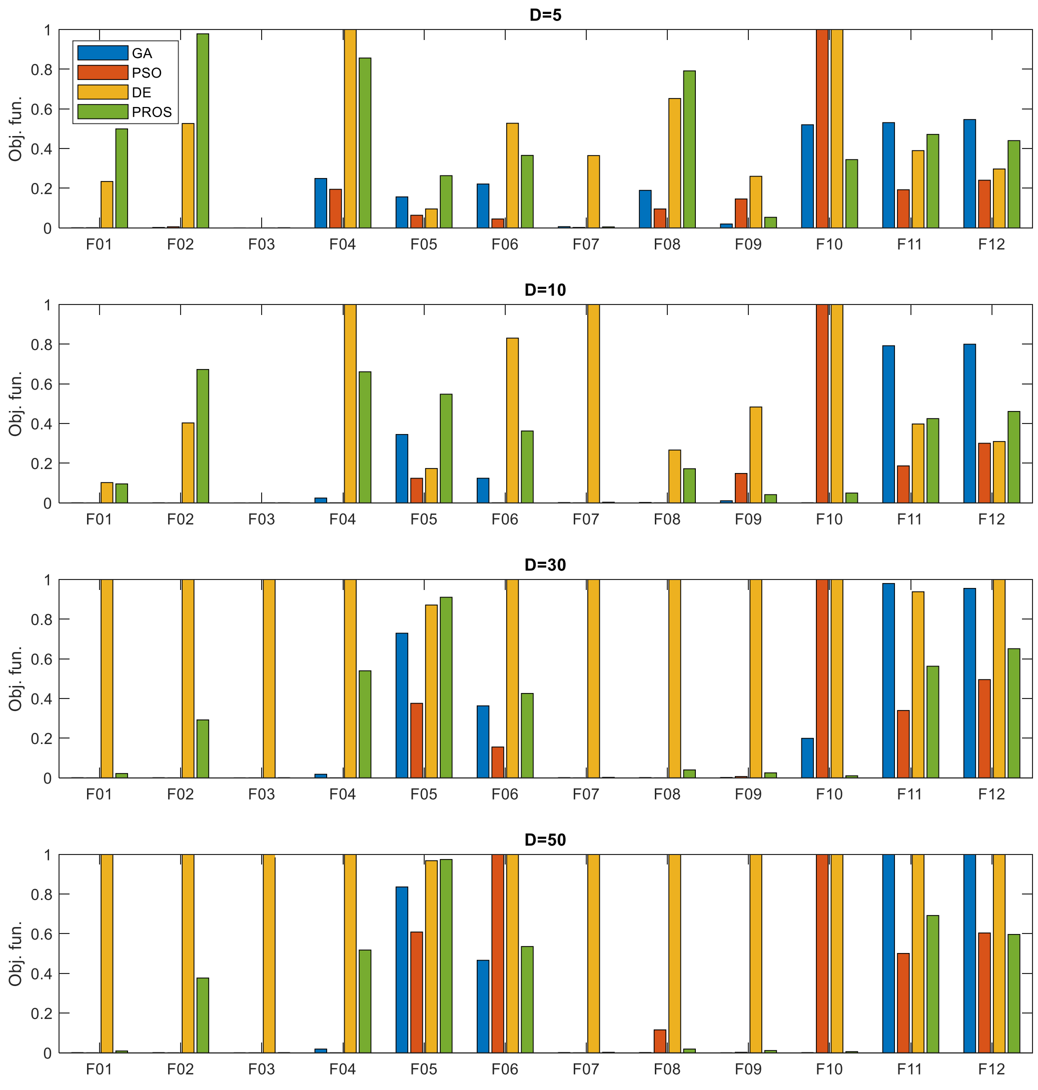


Figure 4. Best results (final values of the objective functions) for the various optimizers for D = 5, 10, 30, and 50, for all 12 objective functions (averages of 10 runs)—lower is better.

3.2. Convergence History for Each Algorithm

The convergence histories of each algorithm, on each problem and for the number of dimensions $D = 5$, $D = 10$, $D = 30$, and $D = 50$ are presented in Figures 5–8, respectively, as averages of 10 runs. The PROS algorithm is shown with a green line. It is clearly shown that the PRS algorithm (purple line) exhibits very slow convergence in all cases. Again, the convergence rate of PROS is fast compared to the other algorithms, with few exceptions.

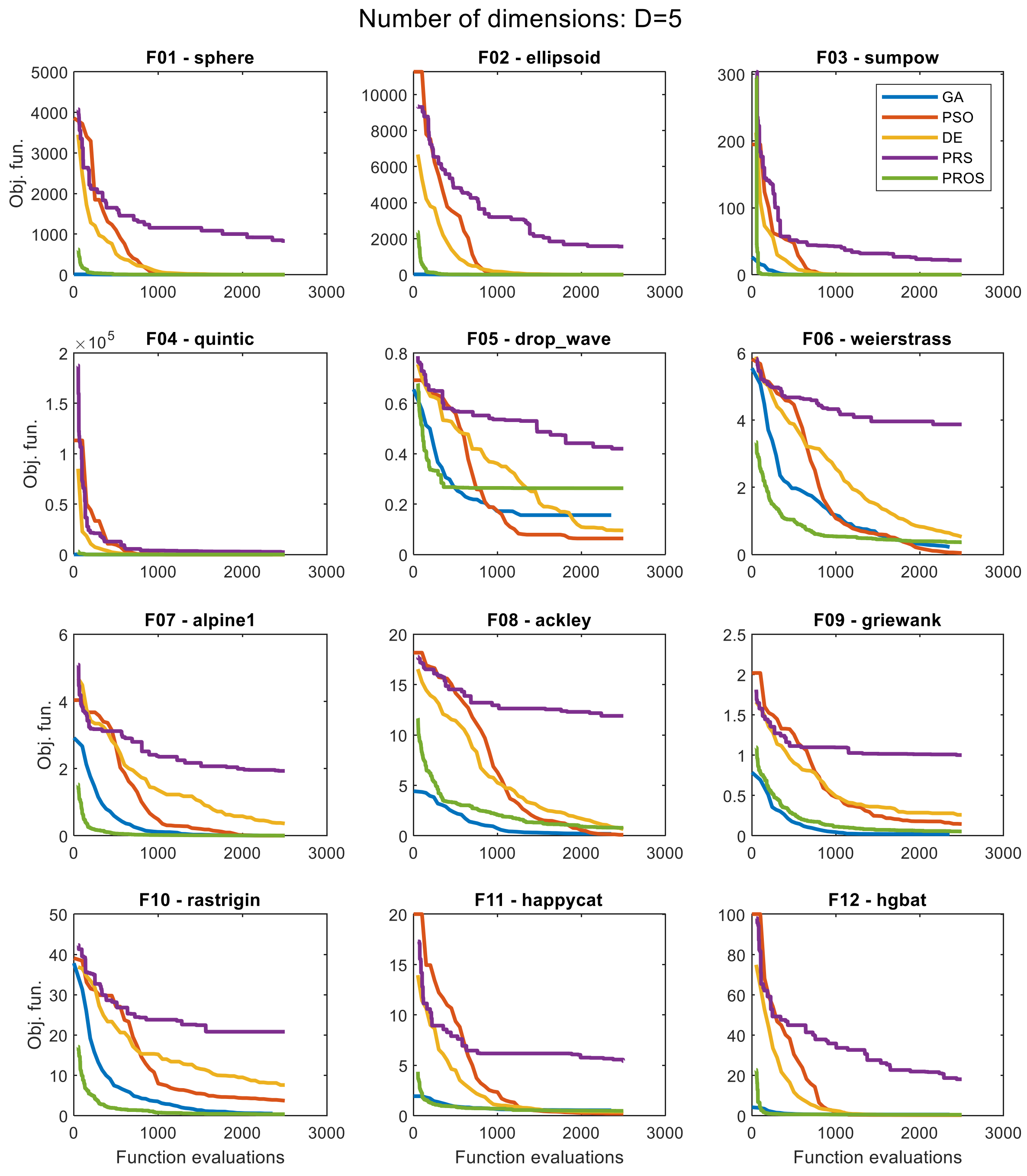


Figure 5. Convergence history of the various optimizers for 5 dimensions ($D = 5$), for all 12 objective functions (average of 10 runs).

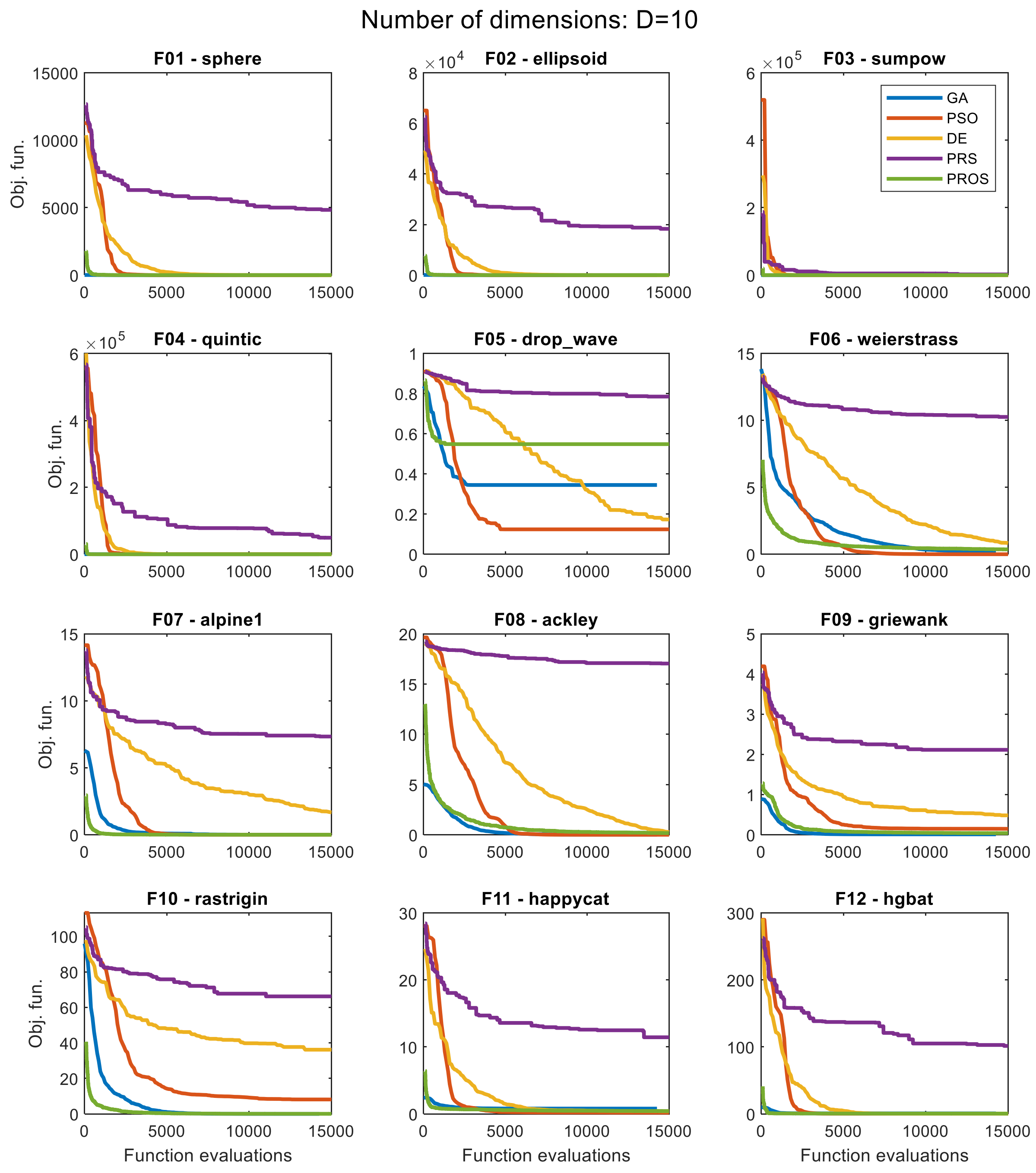


Figure 6. Convergence history of the various optimizers for 10 dimensions (D = 10), for all 12 objective functions (average of 10 runs).

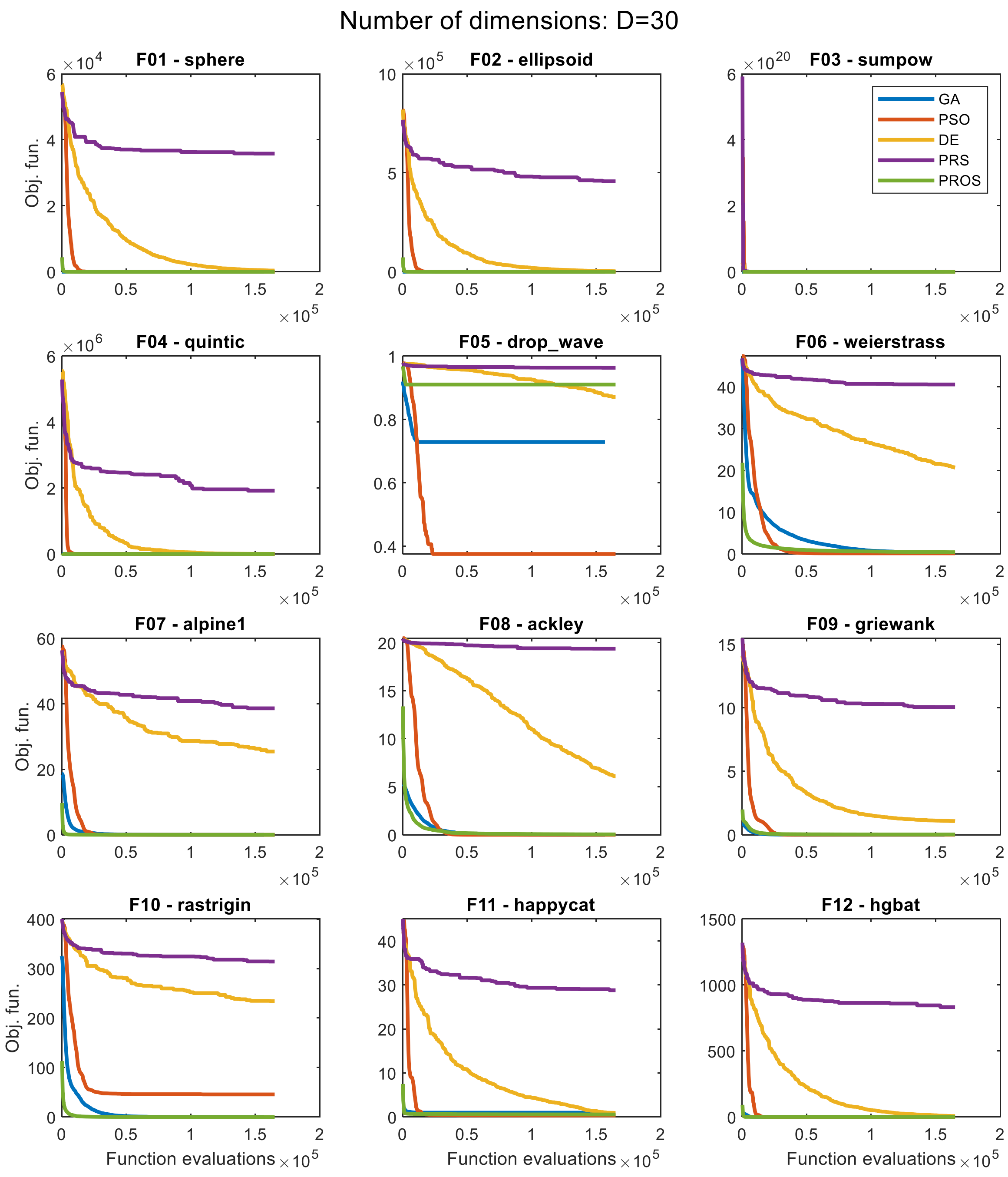


Figure 7. Convergence history of the various optimizers for 30 dimensions ($D = 30$), for all 12 objective functions (average of 10 runs).

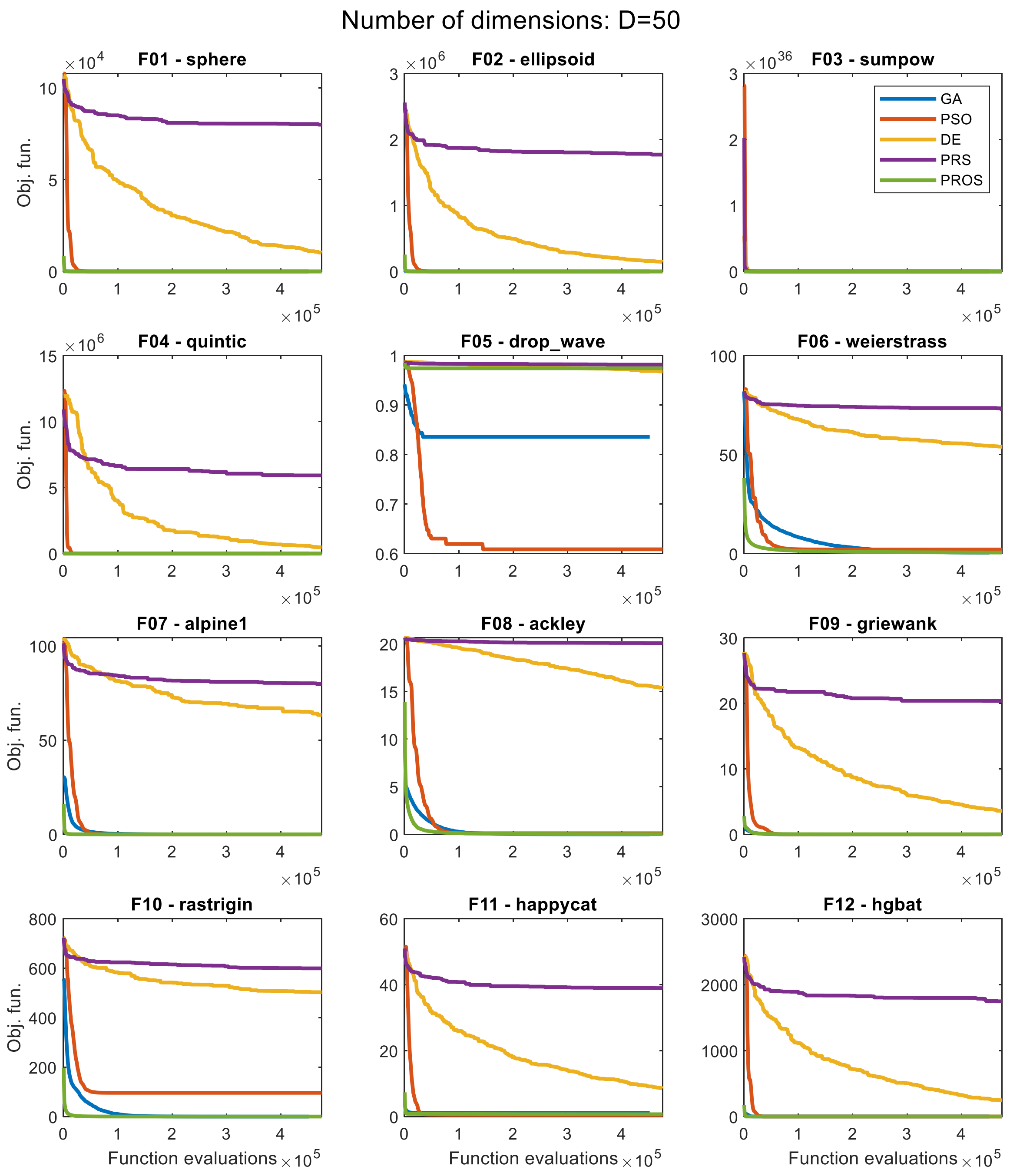


Figure 8. Convergence history of the various optimizers for 50 dimensions ($D = 50$), for all 12 objective functions (average of 10 runs).

3.3. Computational Efficiency

The time for the 10 runs of the optimization procedure was recorded for each algorithm, each function (F01–F12), and each number of dimensions ($D = 5, 10, 30, 50$). Some objective functions are harder to calculate and the corresponding optimization problem also becomes harder and more time consuming. Figure 9 shows the time needed for the optimization of each function, for each method, and for $D = 5, 10, 30, 50$. It is obvious that problem F06 is much harder than the other problems, because of the complexity of this objective function. The time in Figure 9 is in logarithmic scale.

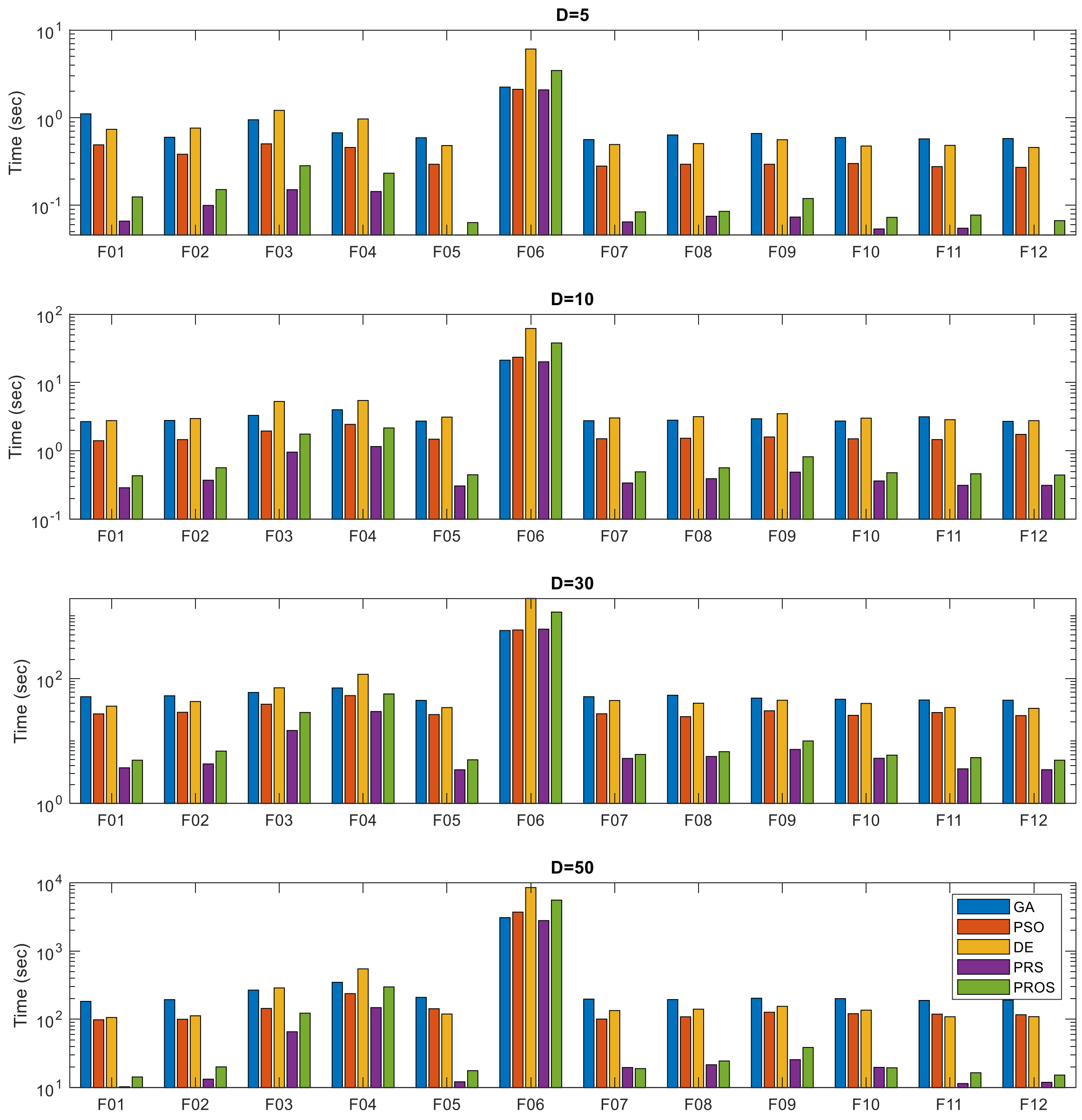


Figure 9. Time needed for the various optimizers, for all 12 objective functions and D = 5, 10, 30, and 50. The y-axis is in logarithmic scale.

The computational time needed by PROS is much smaller than the one needed by GA, PSO, or DE in most of the cases, with the exception of function F06. In addition, it is slightly larger than the one of the PRS method, which, however, shows very poor performance in terms of the final optimum achieved. The PROS method is much faster than all three established optimization algorithms (based on the same number of function evaluations) in almost all the problems, with only very few exceptions.

3.4. Conditions under Which PROS Can Be Trapped in A Local Optimum

Given the formulation of the algorithm, it is obvious that a candidate solution can only move strictly orthogonally in the search space. No diagonal or other kind of movement is allowed, which can lead to the candidate solution being trapped in a local optimum under specific conditions. This behavior was not exhibited in any of the test examples examined. However, theoretically there are specific cases where it can happen if there is no improvement in the value of the objective function in any orthogonal direction. Below we examine such a special case in a two-dimensional problem, where the aim is to minimize the simple two-variable function $f(x)$:

$$\begin{gathered} f(\boldsymbol{x}) = f(x_1, x_2) = x_1 + x_2 - 3x_1x_2 + 1 \\ x_1, x_2 \in [0, 1] \end{gathered} \tag{2}$$

We assume that at a specific point of the search, the current candidate solution lies on the origin of axes, i.e., the point $x_1 = 0$, $x_2 = 0$. The values of the objective function at the four corners of the $[0, 1]^2$ square are the following:

- $f(0, 0) = 1$ (current solution)
- $f(1, 0) = f(0, 1) = 2$
- $f(1, 1) = 0$ (global minimum)

The function is plotted in Figure 10 in 3D (as a surface) and 2D (using contour lines). If the current point is (0, 0) then PROS will be permanently trapped at this position. Indeed, it is $f(x_1,0) > f(0,0)$ for every $x_1 \in [0, 1]$, while $f(0, x_2) > f(0,0)$ for every $x_2 \in [0, 1]$. The current point (0, 0) is a local minimum and, in addition, any change in only one of the x_1 or x_2 variables cannot lead to a better value of the objective function (f). There is no improvement in f in any of the orthogonal x_1 or x_2 directions considered on their own. To find a better value of the objective function, the optimizer would need to move diagonally, but such a movement is not allowed in PROS.

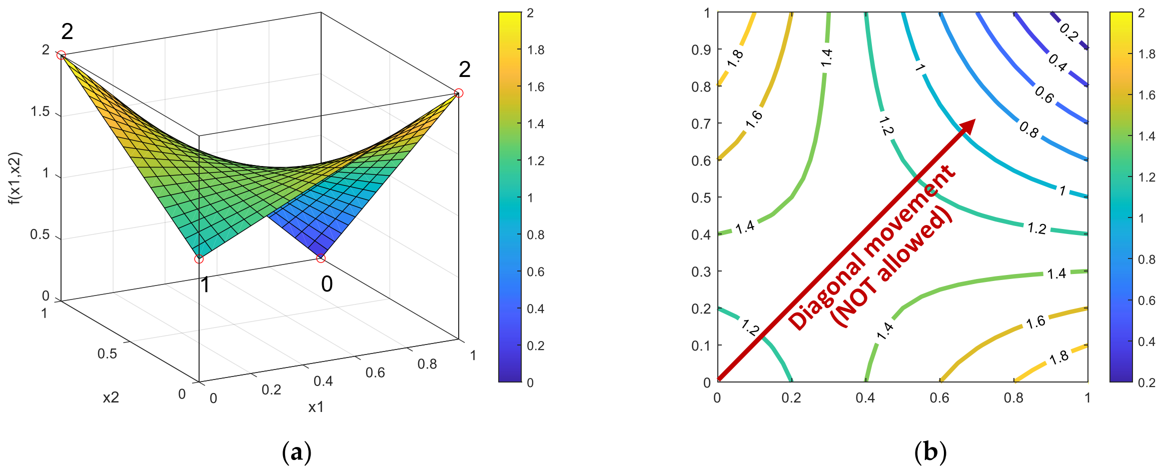


Figure 10. The objective function $f(x_1,x_2) = x_1 + x_2 - 3 \cdot x_1 \cdot x_2 + 1$, plotted in $[0, 1]^2$. PROS is trapped at the local minimum (0, 0) since there is no improvement in f in any of the x_1 or x_2 directions on their own: (**a**); 3D surface plot and (**b**) 2D contour plot.

4. Discussion

In the results section we saw that PROS exhibits very good performance with fast convergence rates and very good final optima, and it is significantly faster than GA, PSO and DE when compared based on the same number of objective function evaluations. In this section we discuss the advantages and disadvantages of PROS and possible applications.

4.1. Advantages and Drawbacks of PROS

The advantages of PROS can be summarized as follows: (i) generality—the algorithm can be applied to virtually any function or engineering problem; (ii) no user-defined parameters—unlike other methods that use parameters such as population size, step size and others, PROS does not have even a single parameter to adjust. This makes the algorithm easy to implement as it can be easily used by non-experts or people who have limited knowledge of the nature of the problem at hand; (iii) easy to program—the method is very easy to code in any programming language. This can significantly reduce the labor cost of an optimization process and it is an appealing feature to both practitioners and theoreticians; (iv) Good convergence rate—the convergence rate of the algorithm is good, showing superior performance when compared to established algorithms in some of the optimization problems; (v) Excellent computational efficiency—due to its simplicity, the implementation of the algorithm is very fast, outperforming the ones of other established algorithms. This means that using PROS one can achieve a larger number of iterations (obj. fun. evaluations) at the same time, in comparison to the other algorithms, when time is the basis of comparison (instead of the number of obj. fun. evaluations as we have done here); (vi) A tool for measuring performance—since PROS is so simple and it has no parameters to adjust, then, according to Occam's razor, any optimization algorithm which is more complex should exhibit clearly superior performance compared to PROS, in most problems. In other words, PROS can serve as a simple comparison tool for measuring the performance of new optimization algorithms in the future.

Local search can be considered a drawback of the method. In many cases, the algorithm appears to converge fast in the beginning, compared to the other algorithms, but not so fast later on. It appears that it has good global search capabilities but somehow limited local search capabilities at the vicinity of the optimum. Another drawback is the possibility of the algorithm being trapped in a local optimum under certain conditions, as discussed in detail in Section 3.4.

4.2. PROS Applications

The proposed optimization algorithm is a powerful modelling and problem-solving methodology, which has a broad range of applications in mathematics, management science, industry, engineering and technology. Due to its simplicity and straight-forward implementation on the computer, it can be easily applied by non-experts, to obtain optimized results in any problem where parameters (design variables) need to be adjusted and a measurable cost or performance function can be formulated.

5. Conclusions

The proposed PROS algorithm is a new, fast, simple and parameterless stochastic optimization strategy that has shown surprisingly positive results despite its simplicity. A total of 12 multi-dimensional mathematical test functions with various levels of complexity have been used to thoroughly test the algorithm. The results indicate that the PROS algorithm achieves significantly good performance and a relatively fast convergence speed when compared with other well-established and more complex optimization methods, such as GA, PSO and DE. In general, it appears that the PROS strategy exhibits good exploration capabilities but shows some difficulties on the exploitation phase. However, the main advantages of PROS are the following: (i) lack of user-defined parameters; (ii) good computational efficiency; and (iii) its exceptional easiness of implementation.

The right usage and correct implementation of optimization algorithms often requires a high level of expertise that can take a long time to be obtained. In contrast, the PROS algorithm requires a minimum level of mathematical knowledge and programming skills while still providing a clear, reliable, elegant and powerful search strategy. Therefore, it can be easily implemented and used even by non-experts in optimization problems, in various applied sciences fields. Additionally, the method can serve as a benchmarking tool for evaluating the performance of other optimization methods. We can claim that if an optimization algorithm "OA" is more complex than PROS but it exhibits poorer performance in the same problem or in a series of selected problems, then according to Occam's razor it can be concluded that OA is not a good alternative since it adds unnecessary complexity without significantly improving performance. Of course, complexity cannot be measured directly or precisely, and it is not easy to be quantified in general. Yet, it still makes sense to claim that in optimization algorithms complexity has to do with (a) the number of parameters that need to be adjusted for the algorithm to work properly in a set of problems; and (b) the difficulty of implementation of the algorithm on a computer code.

Given these criteria, PROS is certainly simpler and less complex than most well-known state-of-the-art optimization methods, yet still effective.

Future Directions

The PROS algorithm was presented in this paper in its simplest form. Based on our research, we believe that there are various ways to further improve the algorithm's performance by searching in a smarter way while keeping the basic search idea. Nevertheless, by doing so, perhaps some extra parameters will need to be added. One idea is to include more search directions when updating the candidate solution. Presently, the search is only in orthogonal principal directions, i.e., the new position of the candidate solution differs only in one of the design variables. Instead of that, the search could follow another, possibly random direction in the design space in each iteration. Another idea is to use a different method when sampling a new position. Based on the experience obtained during the search, we can use another distribution that describes the problem more accurately compared to the uniform distribution used in this work. Alternatively, one could employ a population of agents instead of a single solution in each iteration, rendering the method a population-based method, based on specific rules on how the population is updated in each iteration.

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

Funding: The APC was funded by Oslo Metropolitan University.

Informed Consent Statement: Not applicable.

Data Availability Statement: The full source code of the PROS algorithm as well as the ones of the other optimization methods used in this study, are provided. The code is written in MATLAB (R2020b) and it is available online on GitHub at https://github.com/vplevris/PROS (accessed on 20 May 2021).

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

Appendix A. Nomenclature

Table A1. Acronyms.

Acronym	Meaning
ACO	Ant colony optimization
DE	Differential evolution
GA	Genetic algorithm
HS	Harmony search
IOA	Improved optimization algorithm
NFL	No free lunch
OA	Optimization algorithm
PROS	Pure random orthogonal search
PRS	Pura random search
PSO	Particle swarm optimization

Table A2. Notation and symbols.

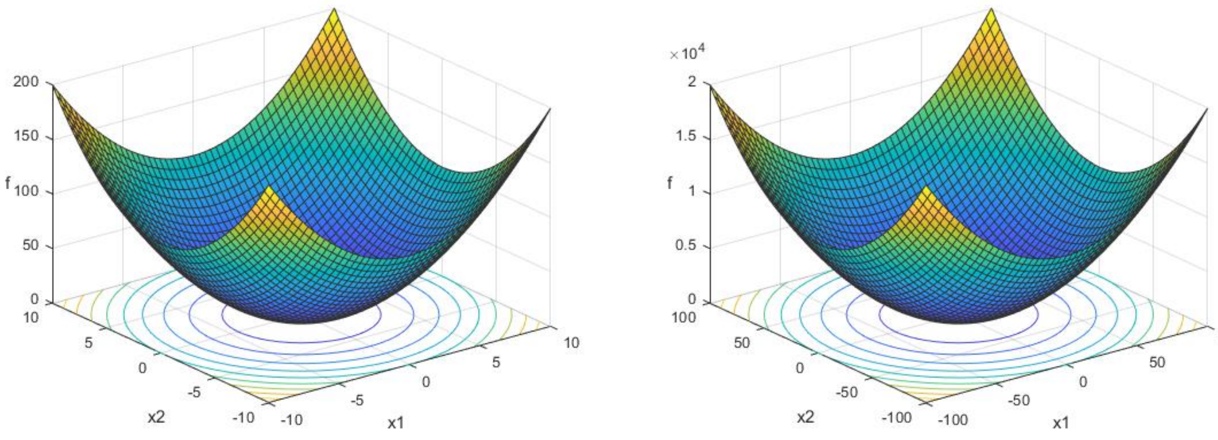
Symbol	Meaning
$f(\boldsymbol{x})$	Objective function
D	Dimensionality of the optimization problem (no. of dimensions)
Ω	Search space
$\mathbb{R}^D$	A real coordinate space of dimension D
$\boldsymbol{x}$ or $\boldsymbol{y}$	vector of decision variables
x_i or y_i	i-th decision variable, i-th element of the decision variables' vector
$\boldsymbol{lb}/\boldsymbol{ub}$	Lower/upper bounds vector
lb_i/ub_i	Lower/upper bound for the decision variable x_i
$\boldsymbol{x}^*$	Decision variables' vector corresponding to the optimum solution
$f(\boldsymbol{x}^*)$	Optimum value of the function f
r	Random number
NP	Population size
$MaxIter$	Maximum number of iterations (or generations)
OFE	Maximum number of function evaluations
F	Differential weight of the DE algorithm
CR	Crossover probability of the DE algorithm

Appendix B. Objective Functions Used in the Study

ID	Name and code name	Search range	Optimum	Properties

F01	Sphere function
	sphere_func
		$[-100, 100]^D$	$f_{01}(\boldsymbol{x}^*) = 0$
			at $\boldsymbol{x}^* = \{0, 0, \ldots, 0\}$
				Unimodal;
				Highly symmetric;
				Rotationally invariant

$$f_{01}(\boldsymbol{x}) = \sum_{i=1}^{D} x_i^2$$

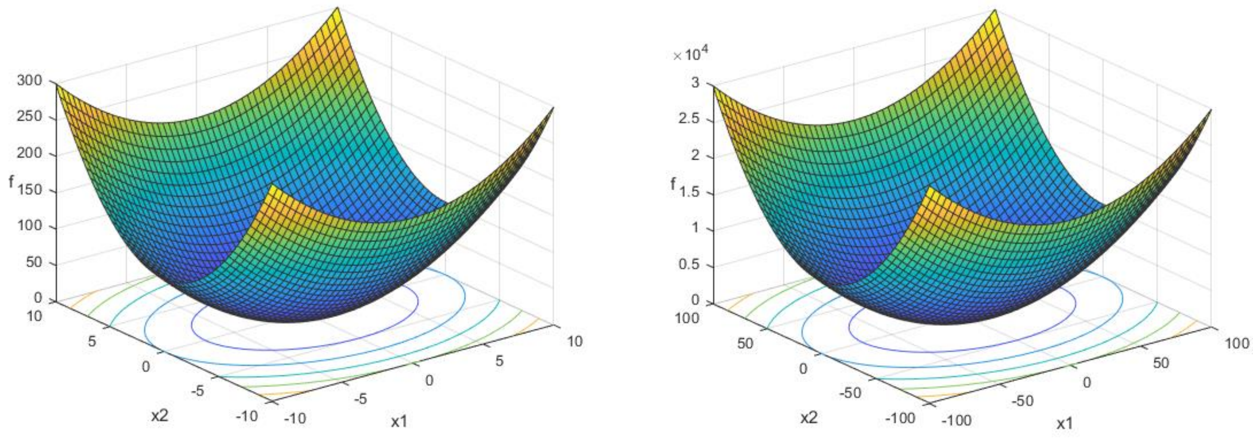


(**a**) $\boldsymbol{x} \in [-10, 10]^2$ (**b**) $\boldsymbol{x} \in [-100, 100]^2$

Figure A1. Sphere function in two dimensions.

ID	Name and code name	Search range	Optimum	Properties
F02	Ellipsoid function ellipsoid_func	$[-100, 100]^D$	$f_{02}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{0, 0, \ldots, 0\}$	Unimodal Symmetric

$$f_{02}(\boldsymbol{x}) = \sum_{i=1}^{D} i \cdot x_i^2$$

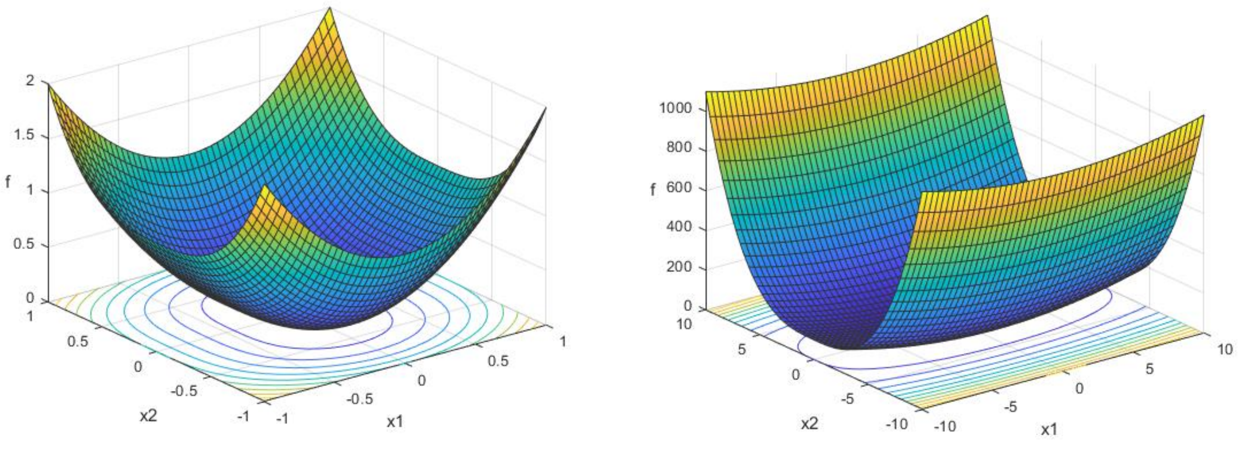


(**a**) $x \in [-10, 10]^2$ (**b**) $x \in [-100, 100]^2$

Figure A2. Ellipsoid function in two dimensions.

ID	Name and code name	Search range	Optimum	Properties
F03	Sum of Different Powers function sumpow_func	$[-10, 10]^D$	$f_{03}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{0, 0, \ldots, 0\}$	Unimodal

$$f_{03}(\boldsymbol{x}) = \sum_{i=1}^{D} \left| x_i \right|^{i+1}$$

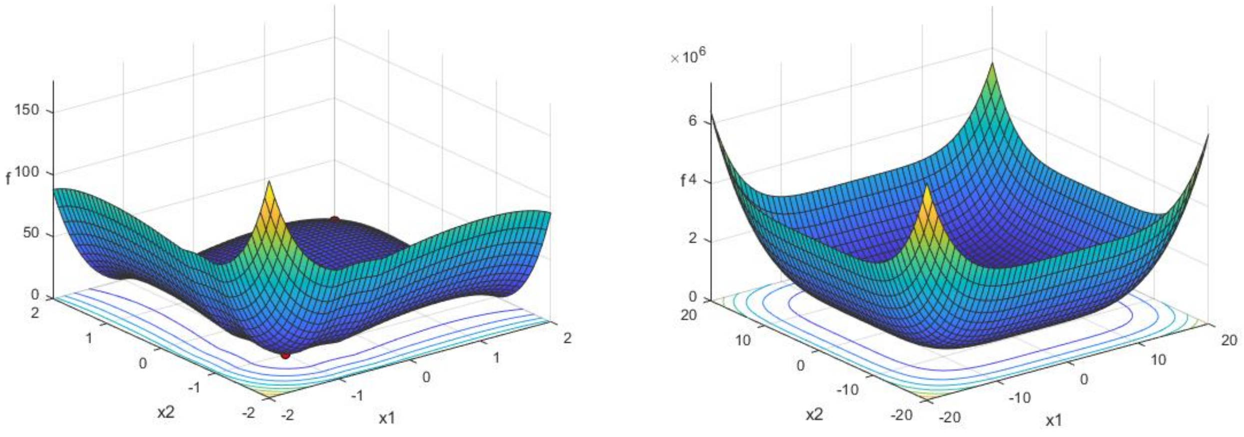


(**a**) $x \in [-1, 1]^2$ (**b**) $x \in [-10, 10]^2$

Figure A3. Sum of different powers function in two dimensions.

ID	Name and code name	Search range	Optimum	Properties
F04	Quintic function quintic_func	$[-20, 20]^D$	$f_{04}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{-1, -1, \ldots, -1\}$ and $\boldsymbol{x}^* = \{2, 2, \ldots, 2\}$	Two global optima

$$f_{04}(\boldsymbol{x}) = \sum_{i=1}^{D} \left| x_i^5 - 3x_i^4 + 4x_i^3 + 2x_i^2 - 10x_i - 4 \right|$$

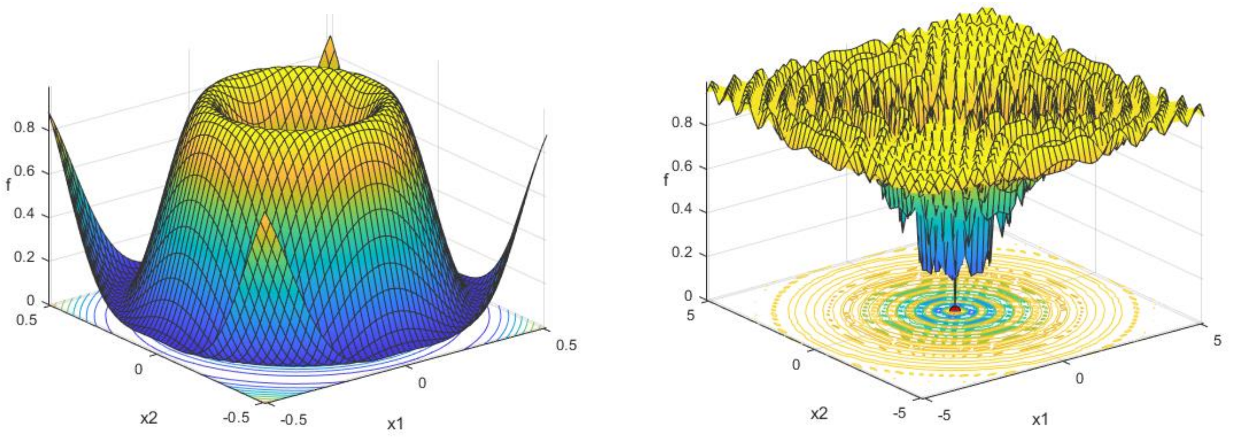


(**a**) $\boldsymbol{x} \in [-2, 2]^2$ (**b**) $\boldsymbol{x} \in [-20, 20]^2$

Figure A4. Quintic function in two dimensions.

ID	Name and code name	Search range	Optimum	Properties
F05	Drop-Wave function drop_wave_func	$[-5.12, 5.12]^D$	$f_{05}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{0, 0, \ldots, 0\}$	Multi-modal Highly complex

$$f_{05}(\boldsymbol{x}) = 1 - \left(1 + \cos\left(12\sqrt{\sum_{i=1}^{D} x_i^2}\right)\right) \Big/ \left(0.5\sum_{i=1}^{D} x_i^2 + 2\right)$$



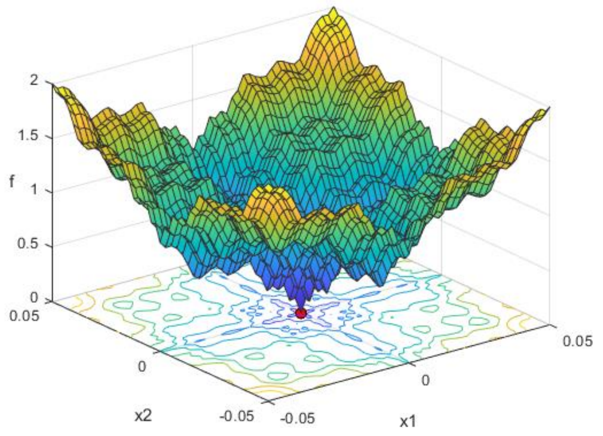
(**a**) $\boldsymbol{x} \in [-0.512, 0.512]^2$ (**b**) $\boldsymbol{x} \in [-5.12, 5.12]^2$

Figure A5. Drop-wave function in two dimensions.

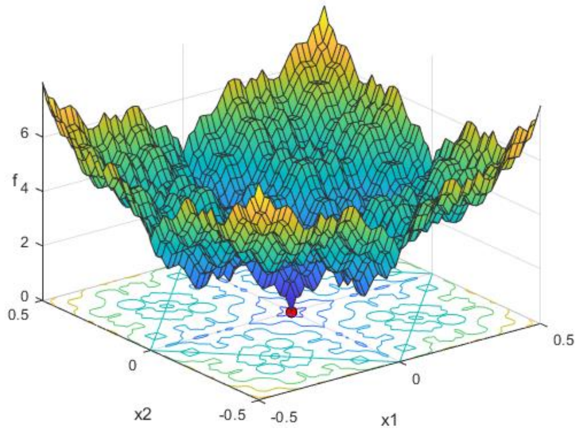
ID	Name and code name	Search range	Optimum	Properties
F06	Weierstrass function weierstrass_func	$[-0.5, 0.5]^D$	$f_{06}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{0, 0, \ldots, 0\}$	Multi-modal Continuous everywhere but only differentiable on a set of points

$$f_{06}(\boldsymbol{x}) = \sum_{i=1}^{D}\left(\sum_{k=0}^{kmax}\left(a^k\cos\left(2\pi b^k\left(x_i+0.5\right)\right)\right)\right) - D\sum_{k=0}^{kmax}\left(a^k\cos\left(\pi b^k\right)\right)$$

$$a = 0.5, \quad b = 3, \quad kmax = 20$$



(**a**) $x \in [-0.05, 0.05]^2$

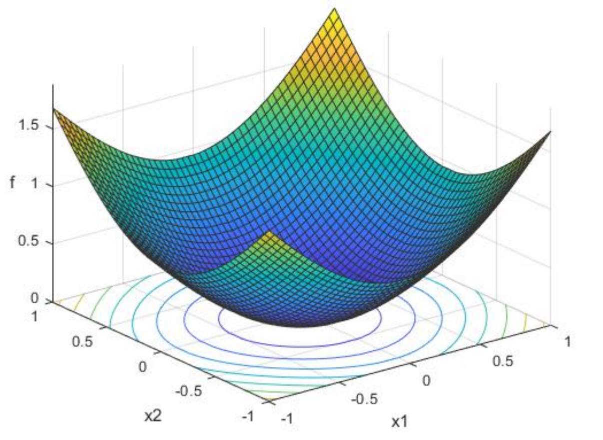


(**b**) $x \in [-0.5, 0.5]^2$

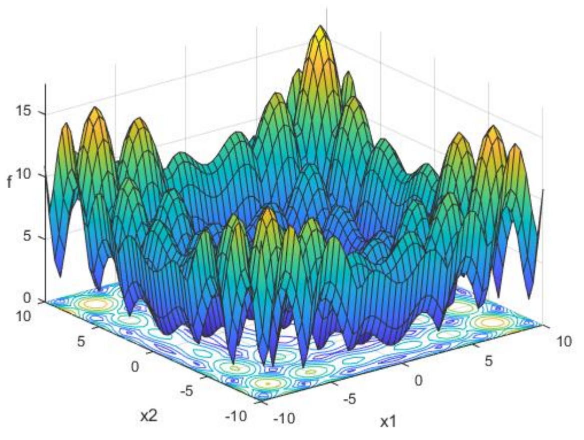
Figure A6. Weierstrass function in two dimensions.

ID	Name and code name	Search range	Optimum	Properties
F07	Alpine 1 function alpine1_func	$[-10, 10]^D$	$f_{07}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{0, 0, \ldots, 0\}$	Multi-modal

$$f_{07}(\boldsymbol{x}) = \sum_{i=1}^{D}\left|x_i\sin\left(x_i\right)+0.1x_i\right|$$



(**a**) $x \in [-1, 1]^2$

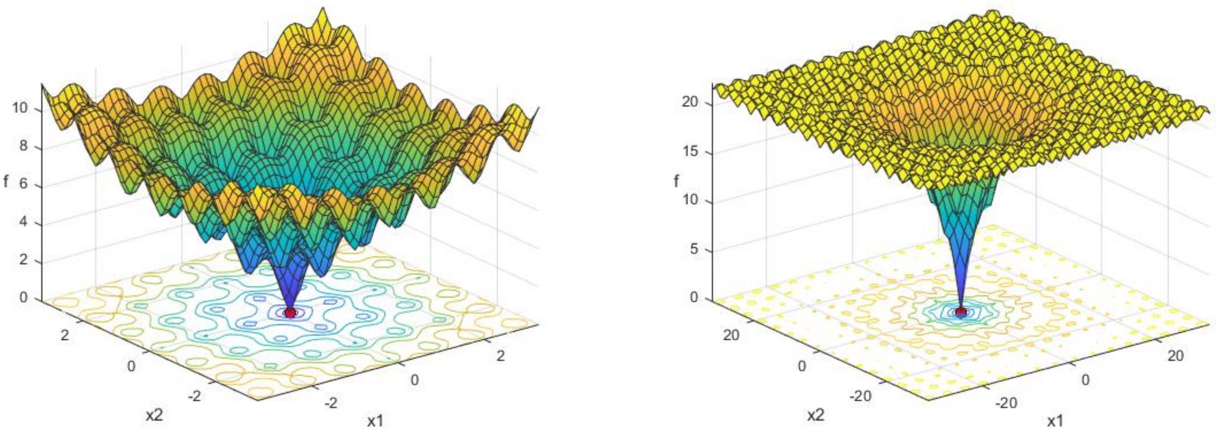


(**b**) $x \in [-10, 10]^2$

Figure A7. Alpine 1 function in two dimensions.

ID	Name and code name	Search range	Optimum	Properties
F08	Ackley's function ackley_func	$[-32.768, 32.768]^D$	$f_{08}(x^*) = 0$ at $x^* = \{0, 0, \ldots, 0\}$	Multi-modal Many local optima with the global optimum in a very small basin

$$f_{08}(\boldsymbol{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D}x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right) + e + 20$$

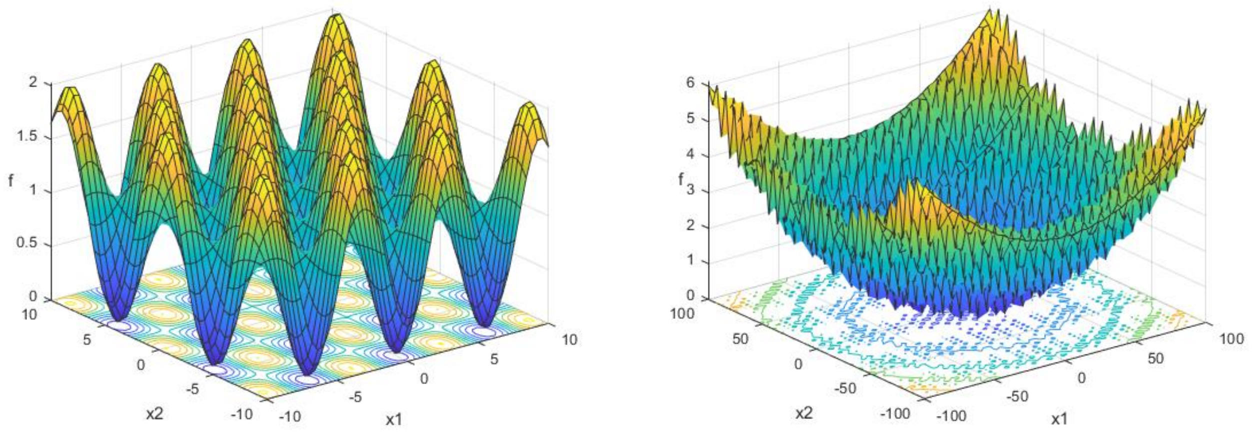


(**a**) $x \in [-3.2768, 3.2768]^2$ (**b**) $x \in [-32.768, 32.768]^2$

Figure A8. Ackley's function in two dimensions.

ID	Name and code name	Search range	Optimum	Properties
F09	Griewank's function griewank_func	$[-100, 100]^D$	$f_{09}(x^*) = 0$ at $x^* = \{0, 0, \ldots, 0\}$	Multi-modal With many regularly distributed local optima

$$f_{09}(\boldsymbol{x}) = \frac{1}{4000}\sum_{i=1}^{D}x_i^2 - \prod_{i=1}^{D}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

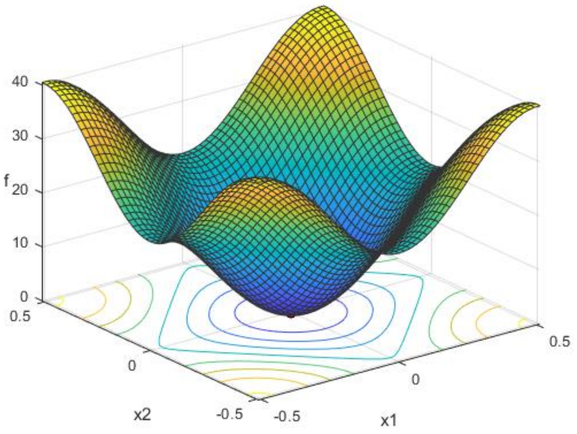


(**a**) $x \in [-10, 10]^2$ (**b**) $x \in [-100, 100]^2$

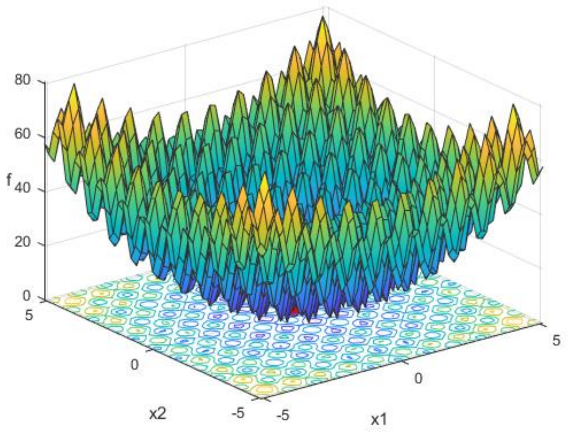
Figure A9. Griewank's function in two dimensions.

ID	Name and code name	Search range	Optimum	Properties
F10	Rastrigin's function rastrigin_func	$[-5.12, 5.12]^D$	$f_{10}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{0, 0, \ldots, 0\}$	Multi-modal With many regularly distributed local optima

$$f_{10}(\boldsymbol{x}) = \sum_{i=1}^{D}\left(x_i^2 - 10\cos\left(2\pi x_i\right)\right) + 10 \cdot D$$



(**a**) $x \in [-0.512, 0.512]^2$

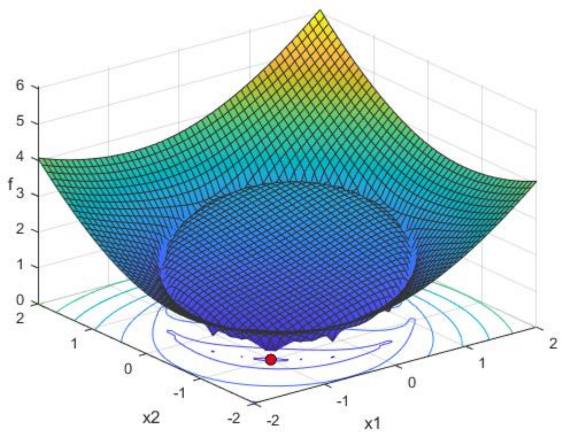


(**b**) $x \in [-5.12, 5.12]^2$

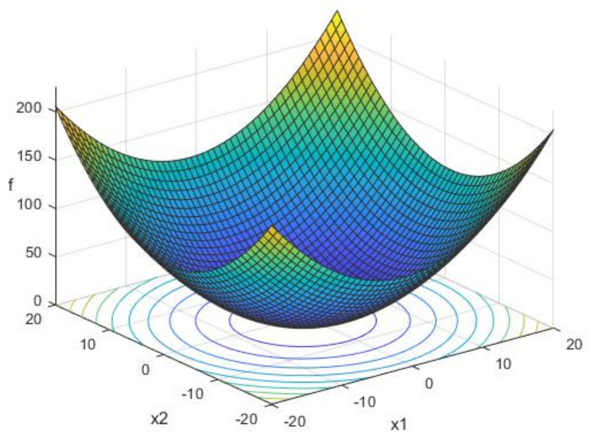
Figure A10. Rastrigin's function in two dimensions.

ID	Name and code name	Search range	Optimum	Properties
F11	HappyCat function happycat_func	$[-20, 20]^D$	$f_{11}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{-1, -1, \ldots, -1\}$	Multi-modal Global optimum located in curved narrow valley

$$f_{11}(\boldsymbol{x}) = \left|\sum_{i=1}^{D} x_i^2 - D\right|^{1/4} + \left(0.5\sum_{i=1}^{D} x_i^2 + \sum_{i=1}^{D} x_i\right) / D + 0.5$$



(**a**) $x \in [-2, 2]^2$

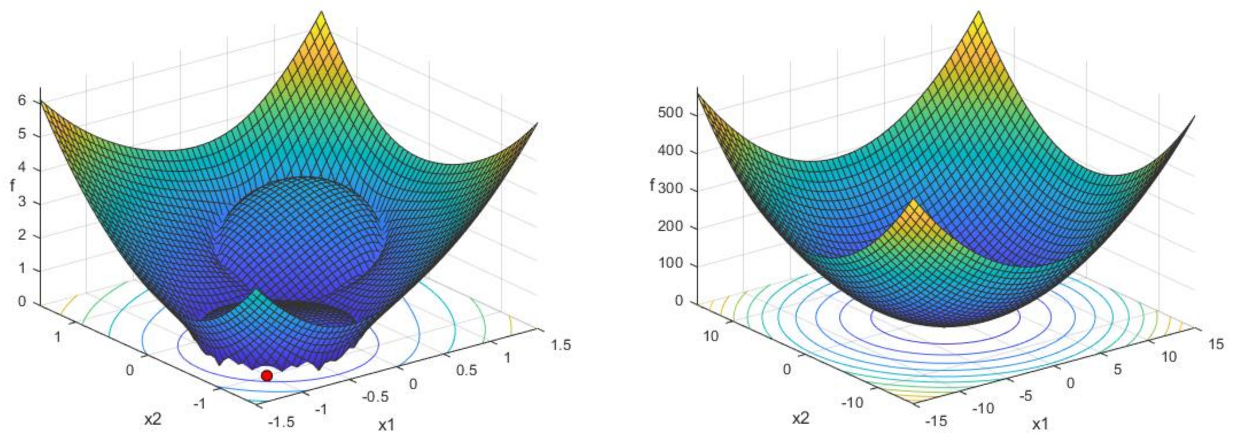


(**b**) $x \in [-20, 20]^2$

Figure A11. HappyCat function in two dimensions.

ID	Name and code name	Search range	Optimum	Properties
F12	HGBat function hgbat_func	$[-15, 15]^D$	$f_{12}(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = \{-1, -1, \ldots, -1\}$	Multi-modal Global optima located in curved narrow valley

$$f_{12}(\mathbf{x}) = \left|\left(\sum_{i=1}^{D} x_i^2\right)^2 - \left(\sum_{i=1}^{D} x_i\right)^2\right|^{1/2} + \left(0.5\sum_{i=1}^{D} x_i^2 + \sum_{i=1}^{D} x_i\right)/D + 0.5$$



(**a**) $x \in [-1.5, 1.5]^2$ (**b**) $x \in [-15, 15]^2$

Figure A12. HGBat function in two dimensions.

References

1. Solorzano, G.; Plevris, V. Optimum Design of RC Footings with Genetic Algorithms According to ACI 318-19. *Buildings* **2020**, *10*, 110. [CrossRef]
2. Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning*; Addison-Wesley Longman Publishing Co.: Boston, MA, USA, 1989.
3. Holland, J. *Adaptation in Natural and Artificial Systems*; University of Michigan Press.: Ann Arbor, MI, USA, 1975.
4. Plevris, V.; Karlaftis, N.D.; Lagaros, N.D. A Swarm Intelligence Approach for. Emergency Infrastructure Inspection Scheduling. In *Sustainable and Resilient Critical Infrastructure Systems: Simulation, Modeling, and Intelligent Engineering*; Gopalakrishnan, K., Peeta, S., Eds.; Springer: Heidelberg, Germany, 2010; pp. 201–230.
5. Plevris, V.; Papadrakakis, M. A Hybrid. Particle Swarm—Gradient Algorithm for Global Structural Optimization. *Comput. Aided Civ. Infrastruct. Eng.* **2011**, *26*, 48–68. [CrossRef]
6. Kennedy, J.; Eberhart, R. Particle Swarm Optimization. In Proceedings of the IEEE International Conference on Neural Networks, Piscataway, NJ, USA, 27 November–1 December 1995; pp. 1942–1948.
7. Georgioudakis, M.; Plevris, V. A Combined Modal Correlation Criterion for Structural Damage Identification with Noisy Modal Data. *Adv. Civ. Eng.* **2018**, *2018*, 318067. [CrossRef]
8. Georgioudakis, M.; Plevris, V. On the Performance of Differential Evolution Variants in Constrained Structural Optimization. *Procedia Manuf.* **2020**, *44*, 371–378. [CrossRef]
9. Georgioudakis, M.; Plevris, V. A Comparative Study of Differential Evolution Variants in Constrained Structural Optimization. *Front. Built Environ.* **2020**, *6*, 1–14. [CrossRef]
10. Storn, R.; Price, K. Differential Evolution—A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *J. Glob. Optim.* **1997**, *11*, 341–359. [CrossRef]
11. Dorigo, M.; Maniezzo, V.; Colorni, A. The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Trans. Syst. Man Cybern. B.* **1996**, *26*, 2941. [CrossRef] [PubMed]
12. Geem, Z.W.; Kim, J.H.; Loganathan, G.V. A New Heuristic Optimization Algorithm: Harmony Search. *SIMULATION* **2001**, *76*, 60–68. [CrossRef]
13. Gao, X.Z.; Govindasamy, V.; Xu, H.; Wang, X.; Zenger, K. Harmony Search Method: Theory and Applications. *Comput. Intell. Neurosci.* **2015**, *2015*, 258491. [CrossRef] [PubMed]

14. Kirkpatrick, S.; Gelatt, C.D.; Vecchi, M.P. Optimization by Simulated Annealing. *Science* **1983**, *220*, 671–680. [CrossRef] [PubMed]
15. Glover, F.; Taillard, E.; Taillard, E. A User's Guide to Tabu Search. *Ann. Oper. Res.* **1993**, *41*, 1–28. [CrossRef]
16. Wolpert, D.H.; Macready, W.G. No Free Lunch Theorems for Optimization. *IEEE Trans. Evol. Comput.* **1997**, *1*, 67–82. [CrossRef]
17. Culberson, J.C. On the Futility of Blind. Search: An. Algorithmic View of "No Free Lunch". *Evol. Comput.* **1998**, *6*, 109–127. [CrossRef] [PubMed]
18. Ho, Y.C.; Pepyne, D.L. Simple Explanation of the No-Free-Lunch Theorem and Its Implications. *J. Optim. Theory Appl.* **2002**, *115*, 549–570. [CrossRef]
19. Koziel, S.; Michalewicz, Z. Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization. *Evol. Comput.* **1999**, *7*, 19–44. [CrossRef] [PubMed]
20. Plevris, V. *Innovative Computational Techniques for the Optimum Structural Design Considering Uncertainties*; National Technical University of Athens: Athens, Greece, 2009; p. 312.
21. Zabinsky, Z.B. Stochastic Adaptive Search for Global Optimization. In *Nonconvex Optimization and Its Applications*; Springer: New York, USA, 2003; ISBN 978-1-4419-9182-9.
22. Spall, J.C. *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*; Wiley: Hoboken, NJ, USA, 2003; ISBN 9780471330523.
23. Brooks, S.H. A Discussion of Random Methods for Seeking Maxima. *Oper. Res.* **1958**, *6*, 244–251. [CrossRef]
24. Peng, J.-P.; Shi, D.-H. Improvement of Pure Random Search in Global Optimization. *J. Shanghai Univ.* **2000**, *4*, 92–95. [CrossRef]