Available online at www.sciencedirect.com
ScienceDirect
Results in Control and Optimization 00 (2023) 1–11
Procedia Computer Science

Analysis of Sequential Parameter Optimization for Computer Simulation Optimization

Andre N. Costa, Joao P. A. Dantas

Decision Support Systems Subdivision, Institute for Advanced Studies, Sao Jose dos Campos - SP 12228-001, Brazil

Abstract

The main goal of this work is to investigate how the Sequential Parameter Optimization (SPO) framework performs if compared with non-sequential approaches. This is of great importance when dealing with expensive black-box functions as is often the case with regards to simulation optimization. For comparison purposes, well-established metaheuristic methods were applied to classical test functions, which, although being very fast to evaluate, pose challenges for their optimization. The idea was to examine whether SPO would outperform the other methods when a smaller amount of function evaluations was utilized, what could be imposed by budget constraints. Black Hole Optimization, Differential Evolution, and Particle Swarm Optimization were the metaheuristics selected in our study, being all populational methods that may require a large number of iterations in order to achieve good results. The conclusion from the conducted analysis shows that SPO, despite being a more expensive method in the computational standpoint, indeed gets to better results when there is a very limited number of function evaluations available.

Keywords:
Optimization, Metaheuristic, Sequential Parameter Optimization

1. Introduction

Experiments have been widely used to study the relationship between a set of inputs to a system and the resulting outputs. However, many physical processes are difficult or impossible to explore directly by conventional experimental methods. This may be due to the large costs that can be involved, or even the complexity of the experimental setting that would be necessary, which could also present excessive risks to the experimenter, or to the system to be experimented itself.

As computing power has increased, it has become possible to model some of these complex processes by sophisticated computer code. In a similar way to physical experiments, one can vary the inputs to these elaborate sets of code and observe how the outputs are affected. This process is traditionally called computer experiments and are becoming increasingly popular surrogates for their physical counterparts [1].

Many of these computer experiments are conducted by simulation, a term that can be defined in several ways throughout the scientific literature. In the context of our work, we adopt the following definition: a simulation model is a mathematical model that is converted into a computer program to be solved by means of experimentation. Therefore,

Email addresses: negraoanc@fab.mil.br (Andre N. Costa), dantasjpad@fab.mil.br (Joao P. A. Dantas)

simulation implies that the analysts do not solve their model through mathematical analysis, instead, they try different values for the input parameters of their model in order to learn what happens to its outputs [2].

The treatment of simulation outputs often involves their optimization, that is, analysts frequently aim to define which is the set of inputs that leads to the optimal (maximum or minimum) values of the outputs. However, usually the size and complexity of the problem prevents a naive application of classical optimization methods. This happens when there is no closed form for the objective function, what is often the case when simulation is used.

Objective function evaluation via simulation is time-consuming, even if considering the recent advances in computing power. Hence, the traditional approaches to capture the behavior of the objective function would require an enormous amount of computer time with the simulation software, which could make impractical the use of simulation-based methods with the goal of optimization. What is needed, therefore, are optimization techniques that utilize a manageable amount of simulation time and produce optimal or near-optimal solutions [3]. For instance, the Traveling Salesman Problem, one of the most studied routing problems within the combinatorial optimization field, even though it comes from a relatively simple idea, finding its solution is NP-Hard, which stimulates the great interest in finding efficient ways to solve it such as employing heuristics to generate approximate solutions [4].

Many optimization techniques are associated with a refined design of experiments to achieve these solutions. When dealing with deterministic simulation, these designs do not take more than one observation at any set of inputs, inasmuch as they would lead to the same results. Moreover, since there is no knowledge with regards to the proper relationship between the response and inputs, i.e., it is a black-box type of system, designs should allow the fitting of a variety of models and provide information about all portions of the experimental region, what may be defined as a space-filling design [1].

This type of design is deemed adequate when the runs of a computer experiment are expensive or time-consuming so that observing the response at a large number of inputs is not possible. [5] proposed the use of a rule-of-thumb for determining a reasonable sample size to use in initial designs that consisted of selecting a sample size of $10d$ when the input space is of dimension d. This rule-of-thumb was conceived with the goal of providing enough points for a more accurate initial fitting of the black-box function analyzed, that is, its metamodel, which are also called response surfaces, surrogates, and emulators.

As defined by [2], a metamodel is an approximation of the input/output (I/O) function that is defined by the underlying simulation model. A recent type of metamodel that is gaining popularity in simulation is called Kriging, which was introduced as a metamodel for simulation models in [6]. One of its advantages over traditional polynomial regression metamodels is that Kriging is a global model, instead of local. It is used for prediction with the final goal of optimization.

A very well-known sequential optimization method called Efficient Global Optimization (EGO), uses a global Kriging metamodel to predict the output of a new point, which is done through the maximization of the Expected Improvement (EI), comparing this new point and the best point that was found so far. Then a new Kriging metamodel is fitted to the old I/O data and the new point I/O. This is done as long as there is still computer budget available [5].

Sequential approaches are widely used in the simulation optimization realm, enabling the analysts to learn about the I/O behavior of the simulated system as data is collected, that is, before having to decide on the next input combination to be simulated. [7] described a framework that generalizes these types of approaches, called Sequential Parameter Optimization (SPO). Although this method was initially conceived for tuning the parameters of optimization methods, it can be used for general optimization as well. It uses the available budget sequentially, i.e., the information from the exploration of the search space guides the search by building a metamodel. Then, it chooses new design points based on predictions from the metamodel, which is refined stepwise to improve knowledge about the search space [7].

Our work aims to compare the use of this framework with traditional optimization techniques, when dealing with a very limited number of possible objective function evaluations. Although the experiments were conducted with very fast test functions, the idea was to check if better results could be reached with less iterations by the use of a sequential approach. It is important to note that classical test functions, also called artificial landscapes, seldomly correspond to real-world problems, since their main objective is to test the optimization methods in most extreme conditions. Thus, the fitting of these functions by metamodels is often hindered by their harsh properties, which can vary in terms of features like modality, basins, valleys, separability, and dimensionality.

The comparison is done through the use of RStudio and LOF, a framework developed by the Brazilian Air Force's Institute of Advanced Studies (IEAv). This framework has implementations of Black Hole Optimization (BHO)

and Particle Swarm Optimization (PSO), as well as the test functions employed (Ackley, Griewank, Sphere, and Zakharov). On the other hand, RStudio was utilized for the configuration of SPO, through a package called SPOT (Sequential Parameter Optimization Toolbox), and for the handling of Differential Optimization, which is also the optimizer of the Kriging model within SPOT.

The paper is organized as follows. Section 2 gives a brief overview of the methods utilized, as well as the test functions selected. Section 3 provides the methodology employed for comparing the optimization methods considered. Section 4 presents the results obtained from the optimization of the benchmark functions. Section 5 summarizes the paper, pointing to the next steps to be taken.

2. Background

This section introduces a series of optimization methods that we used in our comparison, applied to some well-established test functions. Additionally, we present the software tools that provide what is needed for performing the proposed experiments.

2.1. Optimization Methods

Besides the description of the non-sequential optimization methods that we selected for comparison, in this subsection we provide an overview of SPO, as well as a brief characterization of the employed test functions.

2.1.1. Sequential Parameter Optimization (SPO)

SPO is a framework based on active experimentation that aims to test some hypotheses, according to the following procedure [8]:

1. Select a model F (e.g., through kriging) to describe a functional relationship;
2. Select an experimental design, e.g., Latin Hypercube Design (LHD);
3. Generate data, i.e., perform simulation experiments; and
4. Refine the model until the hypothesis can be accepted/rejected.

It was proposed by [7] to efficiently design metaheuritics, being also applicable as an optimization method in itself. This approach has its origins in Design of Experiments (DOE) [9] and in Design and Analysis of Computational Experiments (DACE) [10].

SPO uses the available budget sequentially, i.e., it uses information from the exploration of the search space to guide the search by building a metamodel (e.g. kriging). After that, it chooses new design points based on predictions (e.g. expected improvement) from the metamodel. Thus, it refines the metamodel stepwise to improve knowledge about the search space [11]. Its main goal is to ease the burden of objective function evaluations, when a single evaluation requires a significant amount of resources, which is not the case when testing benchmark functions, but frequently happens in the context of simulation optimization. Metamodel-based optimization is considered to be a relatively efficient optimization method when compared with other optimization techniques [12]. To achieve that, it often relies on balancing the search within the local area of the current optimum and the entire sample space, which may be done sequentially, through the evaluation of the expected improvement, for instance[13].

In summary, the process is done from an initial space-filling design, which, in our case, is the Latin hypercube, accounting for box constraints for the inputs. The information from the exploration is provided by the test functions. The kriging metamodel, combined with EI, allows for sequentially increase the initial design, i.e., after the benchmark functions are evaluated the next input combination is selected. Whereas some combinations are selected to improve the kriging metamodel (global search), some other combinations are added because they seem to be close to the local optimum (local search) [7]. The final knowledge is gathered from the optimization of the latest metamodel, which is done through L-BFGS-B [14], which is a quasi-Newton method that approximates the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm using a limited (L) amount of computer memory. Its B version is an extension to handle simple box constraints on variables. This method can be substituted by another optimization technique, such as Differential Evolution.

2.1.2. Differential Evolution (DE)

DE is an Evolutionary Algorithm (EA), that is, it relies on mutation, recombination and selection to evolve a collection of trial solutions (population) towards an optimal state. It was firstly proposed by [15] for global optimization of continuous and discrete numerical variables.

The individual candidate solutions within DE are called parameter vectors or genomes. The algorithm operates through the same computational steps as employed by a standard EA. However, as a particularity, DE employs the difference of the parameter vectors to explore the objective function landscape. Therefore, its recombination technique involves the creation of new candidate solution components based on the weighted difference between two randomly selected population members added to a third population member, what perturbs population members relative to the spread of the broader population [16].

As mentioned in Section 2.1.1, DE is often utilized in the context of SPO. As an example, [17] demonstrate that particle swarm optimization based approaches were outperformed by differential evolution. If compared to previous results, their approach required only one tenth of the number of function evaluations [7].

2.1.3. Particle Swarm Optimization (PSO)

The PSO is inspired by the social behavior of animals, such as bird flocking and fish schooling. Its algorithms were originally introduced by [18] to find global minimum of functions. They utilize the Reynolds model of flocking [19] to evolve the existing solution generation to a new one until the best value is identified. It initiates from a user-specified number of starting values, called particles, while the collection of all the particles is called a swarm.

Each particle has a certain position and velocity, which is used to obtain the enhancement of the global best solution, what is done when the PSO adds this velocity to the particle position. If the best local solution has a lower fitness value than the fitness of the current global solution, then the best local solution replaces the best global solution [20]. PSO can be easily implemented and it does not require much computational power, such as memory and CPU. Moreover, it does not require gradient information with regards to the objective function, but only function values [1].

2.1.4. Black Hole Optimization (BHO)

BHO is a population-based algorithm that, as done in many other populational methods, generates populations of candidate solutions within the search space. In this particular method, the evolving of the population is done by moving all trial solutions towards the best value in each iteration, which is defined as the black hole, and replacing those candidates that enter within the range of the black hole by newly created candidates in the search space [21].

Therefore, the black hole may change at each iteration, as a new best candidate is found, while the other members of the population are considered as regular stars. Similarly to PSO, the stars move during the optimization process, however, this is done only towards the best solution, acting as an intensification mechanism. The exploration of the search space is done through the rebirth of stars, which often happens at the boundaries of the search space.

2.2. Test Functions

There are many benchmark functions used to evaluate the performance of optimization techniques. We have selected four 2-dimensional functions. Besides, although they are all scalable, differentiable, and continuous, there are some variations between them with regards to modality and separability. Fig. 1 presents a bidimensional plot of these functions, whereas the next section describe some of their main characteristics.

All functions have dimension n and present a minimum value of 0 in its global minimum point, which is always $(0, 0, ..., 0)$. The dominion of the functions are different, being showed in Table 1, which also presents the classification with respect to their modes.

2.2.1. Ackley [22]

The Ackley function is a non-convex function characterized by a nearly flat outer region and a large hole at the center. Its many local minima pose a risk for optimization algorithms to be trapped in, a common feature for multimodal functions. In addition, it is a non-separable function expressed by equation 1.

$$f(x_1, x_2, \ldots, x_n) = -ae^{-b\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(cx_i)} + a + e^1 \tag{1}$$

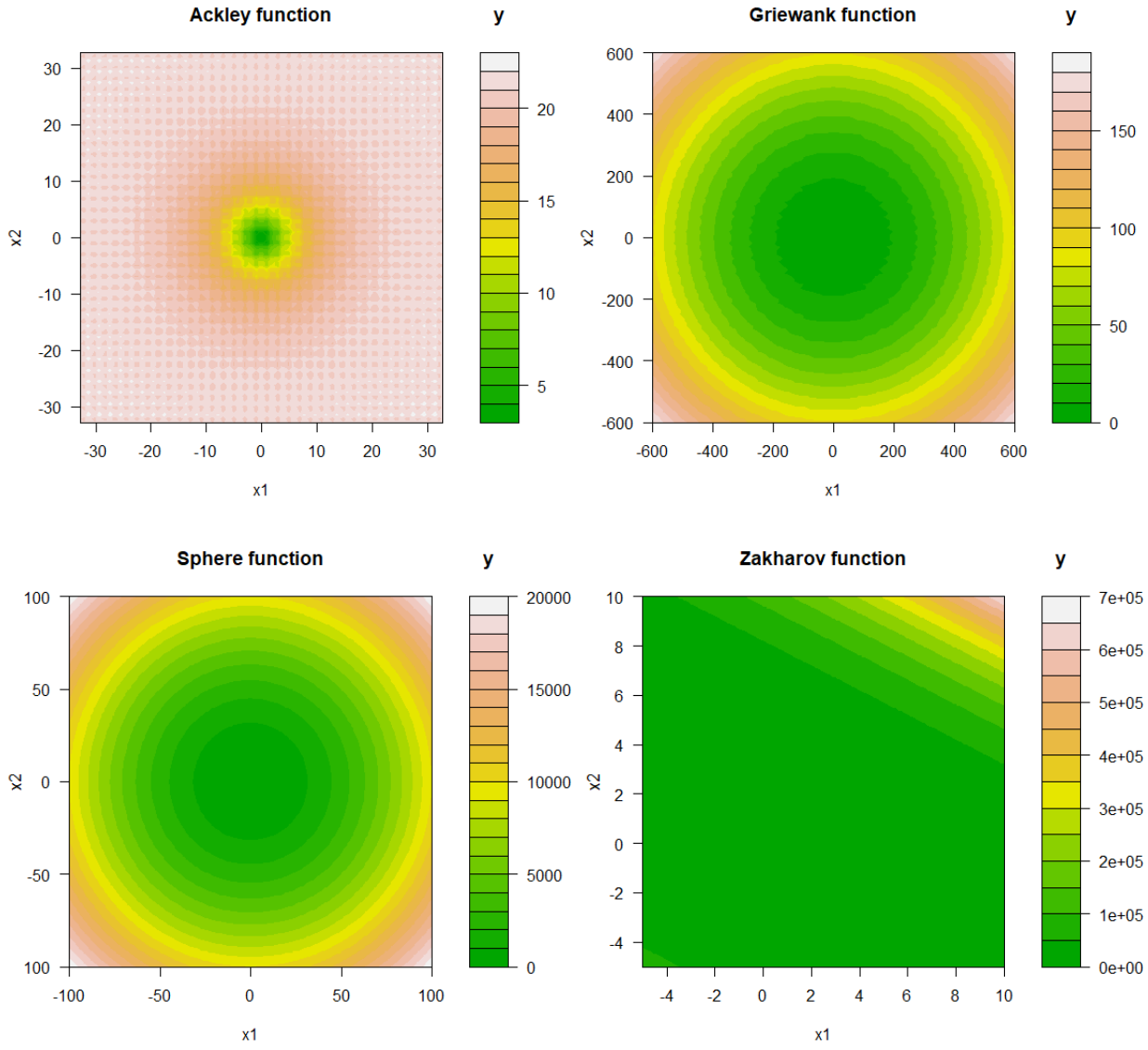

Figure 1. 2D plots of the selected test functions.

Table 1. Dominions and modalities of the test functions.

Name	Dominion	Modality
Ackley	$[-32.768, 32.768]^n$	Multimodal
Griewank	$[-600.0, 600.0]^n$	Multimodal
Sphere	$[-100.0, 100.0]^n$	Unimodal
Zakharov	$[-5.0, 10.0]^n$	Unimodal

2.2.2. Griewank [23]

The Griewank function also has many local minima that are widespread in a regular fashion, what characterizes it as a multimodal function. It is non-separable as well, being defined by equation 2.

$$f(x_1, x_2, \ldots, x_n) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) \tag{2}$$

2.2.3. Sphere

Alternatively, sphere is a unimodal function, as well as separable. It has the simplest mathematical expression when compared with the other benchmark functions, being defined in equation 3.

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} x_i^2 \tag{3}$$

2.2.4. Zakharov [24]

The Zakharov function has no local minima besides the global one, which characterizes it as an unimodal function. It is a non-separable function expressed by equation 4.

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} x_i^2 + \left(\frac{1}{2}\sum_{i=1}^{n} i x_i\right)^2 + \left(\frac{1}{2}\sum_{i=1}^{n} i x_i\right)^4 \tag{4}$$

2.3. Optimization Tools

There are two software solutions that we used to support our experimentation. They are environments that leverages several packages that implement the optimization methods aforementioned in different forms and programming languages.

2.3.1. RStudio

RStudio is an integrated development environment for the statistical computing and graphics language R. It provides a console, syntax-highlighting editor that supports direct code execution, and tools for plotting, history, debugging and workspace management [25]. R packages can be loaded into the environment, making available diverse tools for the most variate purposes, such as the one presented as follows, which were utilized in the context of this work.

- `SPOT [8]:` A set of tools for SPO utilization. It includes surrogate models, optimizers and design of experiment approaches. The main goal is to ease the burden of objective function evaluations, when a single evaluation requires a significant amount of resources.
- `DEoptim [26]:` Implementation of the differential evolution algorithm for global optimization of a real-valued function with a real-valued parameter vector. It is widely used in diverse domains.
- `soobench [27]:` Collection of different single objective test functions useful for benchmarks and algorithm development. The package includes the following functions: Ackley, BBOB 2009, Branon, Ellipsoidal, Goldstein-Price, Griewank, Kotancheck, Mexican hat, Rastrigin, Rosenbrock, Sphere, and Weierstrass.

- `metaheuristicOpt [28]`: A comprehensive implementation of metaheuristic algorithms for continuous optimization. Currently, the package contains the implementations of 21 algorithms: particle swarm optimization, ant lion optimizer, grey wolf optimizer, dragonfly algorithm, firefly algorithm, genetic algorithm, grasshopper optimization algorithm, harmony search algorithm, moth flame optimizer, sine cosine algorithm, whale optimization algorithm, clonal selection algorithm, differential evolution, shuffled frog leaping, cat swarm optimization, artificial bee colony algorithm, krill-herd algorithm, cuckoo search, bat algorithm, gravitational based search, and black hole optimization.

2.3.2. LOF

LOF is the acronym for LEV Optimization Framework, which is a software developed by the Institute for Advanced Studies' Virtual Engineering Lab – *Laboratório de Engenharia Virtual* in Portuguese. Its aim is to provide solution to general computational optimization problems. This is done through a library of metaheuristics for general application, being parametrized through a graphic user interface, which also provides tools for monitoring the results [29].

The framework is utilized to crosscheck the results obtained through RSudio with respect to BHO and PSO, ensuring that the differences in the algorithms' performance are not implementation issues. Besides, the framework provide more data with respect to the optimization execution than what is provided by the packages in R.

3. Test Methodology

With the tools presented in Section 2.3, the testing of the optimization methods to solve the benchmark functions described in Section 2.2 was a rather straightforward effort. With the exception of Zakharov function, all the others were already available on package `soobench`, making any experimentation in R just a matter of properly parametrizing the optimization techniques. Within the LOF framework, all functions were available, as well as BHO and PSO metaheuristics.

The testing consisted on running SPOT, DE, BHO, and PSO with 400 and 100 functions evaluations for all test problems three times. In addition SPOT was run three times with only 20 evaluations, to test the method in such an extreme configuration. All methods were primarily ran in RStudio, with LOF being used to crosscheck the optimal values obtained from BHO and PSO. The inspection of results coming from DE was conducted by the package `metaheuristicOpt` function, since the initial runs were handled by the package `DEoptim`, which is an alternative for SPOT, instead of the L-BFGS-B method utilized.

For crosschecking, the average value for each group of three runs of all possible combinations of metaheuristics and test functions was considered. These values were compared with a single run from the alternative solution. It is important to note that all parameters available for each particular metaheuristic were kept constant throughout their use, with the exception of the number of function evaluations, which was compatible among the same kinds of runs.

The parameters considered for each method were defined as presented in Table2. For a better understanding of the stated parameters, refer to the cited papers with respect to each method or to the package `metaheuristicOpt`.

For the SPO, the initial LHD points varied according to the number of function evaluations considered, what is stated in Table 3.

4. Results and Discussion

In the next sections (4.1, 4.2 and 4.3) we present the obtained results according to the R package utilized to calculate them. After that, the results for crosschecking are presented in section 4.4, whereas the comparison of the results is made in section 4.5.

4.1. BHO and PSO results (`metaheuristicOpt`)

Table 4 outlines the average results – from the three runs performed – obtained in RStudio through the use of its package `metaheuristicOpt`, which has implementations of the BHO and the PSO methods. When considering the same number of function evaluations, BHO presented better results than PSO for all benchmark functions considered, with the exception of Zakharov, when 100 points were evaluated. For Griewank and Sphere functions, BHO outperformed PSO even when evaluating four times less objective functions.

Table 2. Parameters for the optimization methods.

Method	Parameter	Value
BHO	Population	20
PSO	Population	20
	Inertia weight	0.729
	Group cognitive	1.49445
	Individual cognitive	1.49445
	Maximum velocity	2
DE	Population	20
	Mutation	0.8
	Crossover	0.5
	Strategy	DE/local-to-best/1/bin
SPO	Target	Expected Improvement
	Model	DACE Kriging

Table 3. LHD points for SPO runs.

Function evaluations	LHD points
400	100
100	25
20	5

4.2. DE results (`DEoptim`)

The results from the DE method, implemented in R by 'DEoptim' package, are showed in Table 5. As expected, increasing the number of function evaluations proved to reach better results. However, the differences between 400 and 100 are only from 1 to 3 orders of magnitude.

4.3. SPO results (`SPOT`)

SPO presented similar results if compared to the other methods presented so far, as Table 6 displays. A very noticeable fact is that, in all cases with 100 function evaluations, SPO provided the best result. On the other hand, when 400 evaluations are performed, SPO lies behind almost every other method. Interestingly, if we compare SPO with 400 evaluations with SPO with 100 evaluations, the optimal values are very similar to each other, even though the computational time to achieve the larger number is way longer than the other.

Fig. 2 depicts the SPO method evolution for 400 evaluations applied to all four benchmark functions considered.

4.4. Crosscheck

As Table 6 demonstrates, for most of the cases, results obtained by LOF were better than the ones from RStudio. This may indicate that the implementation was optimized, which could also be influenced by the programming language utilizes – C++ in LOF and R in RStudio. DE results were very similar with each other, which was also expected since they are just different R implementations.

4.5. Comparison

The results presented in the previous sections point to a very satisfactory performance of the SPO method. With a reduced computational budget, such as the 100 evaluations represented, SPO outperformed the other methods up to four orders of magnitude, that is a result 1000 times more precise. It is interesting to state that, for a larger number

Table 4. BHO and PSO results.

Evaluations	**400**		**100**	
Function	**BHO**	**PSO**	**BHO**	**PSO**
Ackley	2.00E-02	2.56E-01	1.86E+00	6.16E+00
Griewank	7.65E-02	4.06E+00	3.15E-01	4.16E+00
Sphere	3.34E-01	9.02E+01	5.73E+00	6.00E+02
Zakharov	4.12E-06	7.85E-04	1.36E-01	7.03E-02

Table 5. DE results.

	Evaluations	
Function	**400**	**100**
Ackley	1,87E-01	7,36E+00
Griewank	6,76E-02	9,05E-01
Sphere	1,68E-02	2,90E+01
Zakharov	1,88E-04	6,57E-02

of evaluations, SPO, besides being much slower, provides worst results for practically all benchmark functions if compared to any of the other methods.

Another interesting comparison was made between SPO with 400 and 100 evaluations. The results obtained were very close to each other, that is, the higher number of iterations did not provide much improvement in the results achieved. However, when only 20 evaluations were performed, the quality of the results decreased significantly, although being still fairly close to the results obtained with 100 evaluations of the other methods.

5. Conclusions

This work aimed to verify if SPO would present itself as a proper alternative to classical optimization methods when the studied objective function is expensive and there is a restricted computational budget available. As the results showed, for a reduced number of function evaluations, SPO was indeed the best option, providing better results in all cases that had only 100 iterations.

It is also important to note that SPO has a more flexible budget management, since it is a sequential method, which can be of great advantage in the context of simulation optimization. If the analyst understands that a higher number

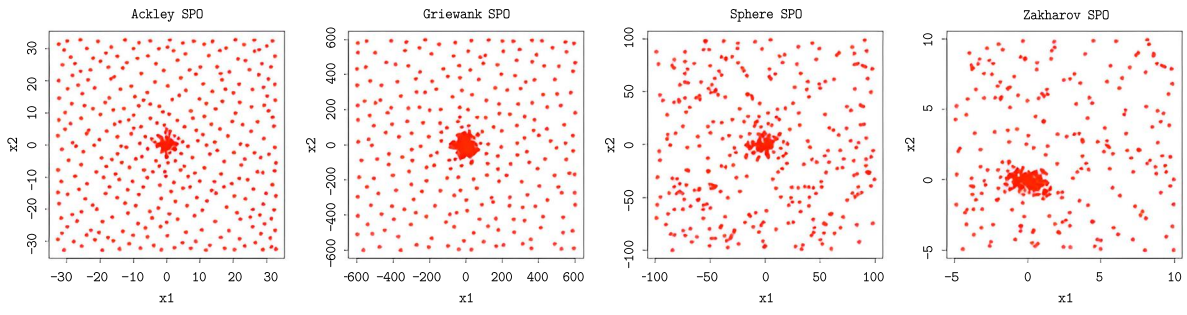

Figure 2. SPO method evolution (400 evaluations).

Table 6. DE results.

	Evaluations		
Function	**400**	**100**	**20**
Ackley	1,89E-01	7,96E-03	5,44E+00
Griewank	2,16E-01	2,16E-01	1,45E+00
Sphere	2,03E-02	7,42E-02	6,67E+01
Zakharov	1,03E-02	1,74E-02	2,31E+00

of function evaluations is needed, SPO is easily extensible to include complementary points.

In summary, although optimization methods may perform in very different ways according to the problem to be solved, this work shows that a specific method may better cope with some of the constraints that may exist, when the evaluated function is too costly. Further studies could include other sequential methods for comparison purposes, which could perform even better than SPO.

References

[1] T. J. Santner, B. J. Williams, W. I. Notz, Space-filling designs for computer experiments, in: The design and analysis of computer experiments, Springer, 2018, pp. 145–200.
[2] J. P. Kleijnen, Design and analysis of simulation experiments, in: International Workshop on Simulation, Springer, 2015, pp. 3–22.
[3] A. Gosavi, Control optimization with reinforcement learning, in: Simulation-Based Optimization, Springer, 2015, pp. 197–268.
[4] J. P. Dantas, A. N. Costa, M. R. Maximo, T. Yoneyama, Enhanced self-organizing map solution for the traveling salesman problem, in: Anais do XVIII Encontro Nacional de Inteligência Artificial e Computacional, SBC, 2021, pp. 799–802.
[5] D. R. Jones, M. Schonlau, W. J. Welch, Efficient global optimization of expensive black-box functions, Journal of Global optimization 13 (4) (1998) 455–492.
[6] J. Sacks, W. J. Welch, T. J. Mitchell, H. P. Wynn, Design and analysis of computer experiments, Statistical science 4 (4) (1989) 409–423.
[7] T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuss, Experimental methods for the analysis of optimization algorithms, Springer, 2010.
[8] T. Bartz-Beielstein, Spot: An r package for automatic and interactive tuning of optimization algorithms by sequential parameter optimization, arXiv preprint arXiv:1006.4645.
[9] J. Antony, Design of experiments for engineers and scientists, Elsevier, 2014.
[10] T. Simpson, V. Toropov, V. Balabanov, F. Viana, Design and analysis of computer experiments in multidisciplinary design optimization: a review of how far we have come-or not, in: 12th AIAA/ISSMO multidisciplinary analysis and optimization conference, 2008, p. 5802.
[11] T. Bartz-Beielstein, C. W. Lasarczyk, M. Preuß, Sequential parameter optimization, in: 2005 IEEE congress on evolutionary computation, Vol. 1, IEEE, 2005, pp. 773–780.
[12] E. Tekin, I. Sabuncuoglu, Simulation optimization: A comprehensive review on theory and applications, IIE transactions 36 (11) (2004) 1067–1081.
[13] N. Quan, J. Yin, S. H. Ng, L. H. Lee, Simulation optimization via kriging: a sequential search using expected improvement with computing budget constraints, Iie Transactions 45 (7) (2013) 763–780.
[14] C. Zhu, R. H. Byrd, P. Lu, J. Nocedal, Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization, ACM Transactions on mathematical software (TOMS) 23 (4) (1997) 550–560.
[15] R. Storn, K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, Journal of global optimization 11 (4) (1997) 341–359.
[16] S. Das, P. N. Suganthan, Differential evolution: A survey of the state-of-the-art, IEEE transactions on evolutionary computation 15 (1) (2010) 4–31.
[17] J. Ziegenhirt, T. Bartz-Beielstein, O. Flasch, W. Konen, M. Zaefferer, Optimization of biogas production with computational intelligence a comparative study, in: IEEE Congress on Evolutionary Computation, IEEE, 2010, pp. 1–8.
[18] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of ICNN'95-international conference on neural networks, Vol. 4, IEEE, 1995, pp. 1942–1948.
[19] C. Reynolds, Boids background and update, http://www. red3d. com/cwr/boids/.
[20] A. Machmudah, S. Parman, Feasible joint angle continuous function of robotics arm in obstacles environment using particle swarm optimization, in: Handbook of Optimization, Springer, 2013, pp. 1047–1071.
[21] A. Hatamlou, Black hole: A new heuristic optimization approach for data clustering, Information sciences 222 (2013) 175–184.
[22] D. Ackley, A connectionist machine for genetic hillclimbing, Vol. 28, Springer Science & Business Media, 2012.
[23] A. O. Griewank, Generalized descent for global optimization, Journal of optimization theory and applications 34 (1) (1981) 11–39.
[24] V. E. Zakharov, Stability of periodic waves of finite amplitude on the surface of a deep fluid, Journal of Applied Mechanics and Technical Physics 9 (2) (1968) 190–194.
[25] J. S. Racine, Rstudio: a platform-independent ide for r and sweave (2012).

[26] K. Mullen, D. Ardia, D. L. Gil, D. Windover, J. Cline, Deoptim: An r package for global optimization by differential evolution, Journal of Statistical Software 40 (6) (2011) 1–26.
[27] O. Mersmann, B. Bischl, soobench: Single objective optimization benchmark functions, URL http://cran. r-project. org/package= soobench. R package version (2012) 1–0.
[28] L. S. Riza, E. P. Nugroho, et al., Metaheuristicopt: An r package for optimisation based on meta-heuristics algorithms., Pertanika Journal of Science & Technology 26 (3).
[29] C. da Silva Junior, W. Saba, N. Abe, A. Passaro, A metric to assist the selection of the particle swarm optimization parameters, Engineering Optimization 2014 (2014) 105.